Douglas Hiura Longo

CENÁRIOS DE USUÁRIO POR MEIO DE DIAGRAMAS DE INTERAÇÃO COM O USUÁRIO

Dissertação submetida ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Santa Catarina para a obtenção do Grau de Mestre em Ciências da Computação.

Orientadora: Prof.ª Dra. Patrícia Vilain

Florianópolis
2015

Ficha de identificação da obra elaborada pelo autor,
através do Programa de Geração Automática da Biblioteca Universitária da UFSC.

Longo, Douglas Hiura
CENÁRIOS DE USUÁRIO POR MEIO DE DIAGRAMAS DE INTERAÇÃO COM O USUÁRIO / Douglas Hiura Longo ; orientadora, Patrícia Vilain - Florianópolis, SC, 2015.
107 p.

Dissertação (mestrado) - Universidade Federal de Santa Catarina, Centro Tecnológico. Programa de Pós-Graduação em Ciência da Computação.

Inclui referências

1. Ciência da Computação. 2. US-UID. 3. ATDD. 4. História de Usuário. 5. Diagrama de Interação do Usuário. I. Vilain, Patrícia . II. Universidade Federal de Santa Catarina. Programa de Pós-Graduação em Ciência da Computação. III. Título.

Douglas Hiura Longo

CENÁRIOS DE USUÁRIO POR MEIO DE DIAGRAMAS DE INTERAÇÃO COM O USUÁRIO

Esta Dissertação foi julgada adequada para obtenção do Título de "Mestre em Ciências da Computação", e aprovada em sua forma final pelo Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Santa Catarina.

Florianópolis, 16 de julho de 2015.

_____
Prof. Ronaldo Dos Santos Mello, Dr.
Coordenador do Curso

Banca Examinadora:

_____
Prof.ª Patrícia Vilain, Dr.ª
Orientadora
Universidade Federal de Santa Catarina

_____
Prof. Raul Sidnei Wazlawick, Dr.
Universidade Federal de Santa Catarina

_____
Prof. Ricardo Pereira E Silva, Dr.
Universidade Federal de Santa Catarina

_____
Prof. Marcelo Soares Pimenta, Dr.
Universidade Federal do Rio Grande do Sul

Em memórias de meus "nonos e nonas": Antônio Dalazen, Santina Martini, José Longo e Alice Sordi.

AGRADECIMENTOS

À professora Patrícia pela valiosa orientação e por ter acreditado, desde o início, no meu trabalho.

À banca, formada pelos Professores Raul, Ricado e Marcelo, pelas importantes orientações sobre o trabalho.

À minha família, em especial à minha mãe Marilene, meu pai Ari e meu irmão Willian, pelo apoio dado durante toda a minha formação acadêmica.

Aos amigos Beatriz, Gustavo, Bruno, Carol, Douglas, Tatiane, por compartilharem um tempo agradável.

À equipe e-Tec, Jhon, André, Renata, Marina, Lucas, Ademar, Fernanda, Jhonatan, Wesly, Junior, Juliana e Cislaghi pela colaboração na realização da pesquisa.

À Professora Silvia, pelas orientações sobre estatísticas e seu amor e dedicação especial.

Aos participantes do experimento, pela colaboração na realização do experimento.

Ao Departamento de Informática e Estatística da UFSC pelo apoio à realização deste trabalho.

RESUMO

O Desenvolvimento Dirigido por Testes de Aceitação (ATDD – *Acceptance Test-Driven Development*) requer que os requisitos da aplicação sejam expressos em formatos executáveis, criados *a priori* do desenvolvimento do código de aplicação. Esses testes devem conter as expectativas dos usuários e são adotados para promover a comunicação e colaboração entre usuários e desenvolvedores. Os requisitos normalmente são derivados de histórias de usuários. Contudo, devido à dificuldade dos usuários em expressarem os requisitos, os desenvolvedores são responsáveis pela especificação e automatização dos testes. Com objetivo de viabilizar a especificação de requisitos por usuários, esta Dissertação propõe os Cenários de Usuários por meio de Diagramas de Interação do Usuário (US-UID – *User Scenarios through User Interaction Diagrams*). Sendo assim, este estudo aborda: a criação de US-UIDs por usuários não técnicos e a execução automatizada dos US-UIDs. Este trabalho foi avaliado por meio de um estudo de caso de um jogo, que resultou em 67% dos requisitos completos e 90% dos requisitos corretos. A partir dos resultados obtidos na avaliação, apresentou-se e implementou-se um *framework* para executar os US-UIDs. Além disto, apresentou-se um estudo de caso baseado em um sistema *web* para verificar a capacidade de rastreabilidade de falha, erro e sucesso do código de aplicação.

Palavras-chave: US-UID. Diagrama de Interação do Usuário. UID. ATDD. Testes Automatizados. FitNesse. História de Usuário.

ABSTRACT

Acceptance Test-Driven Development (ATDD) requires that application requirements to be expressed in executable formats and created a priori the development of the application code. These tests must include the users expectations and are adopted to promote communication and collaboration between users and developers. The requirements are usually derived from user stories. However, customers or users expressing requirements can be difficult. Thus, the developers are responsible for the specification and automation of tests. In order to facilitate the user's requirements specification, this dissertation proposes the User Scenarios through User Interaction Diagrams (US-UID). This study addresses: creation of US-UIDs by non-technical users and automated execution of US-UIDs. This study evaluated through a case study of a game, which resulted in 67% of complete requirements and 90% of correct requirements. Through the evaluation results, we developed a framework to automate the US-UIDs execution. In addition, we show a case study on a Web-base system to check the ability of traceability: failure, error and success of the application code.

Keywords: US-UID. User Interaction Diagram. UID. ATDD. Automated Tests. FitNesse. User Story.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE ABREVIATURAS E SIGLAS

ATDD	Desenvolvimento Guiado por Testes de Aceitação (Acceptance Test-Driven Development)
AAT	Automação de Testes de Aceitação (Automated Acceptance Tests)
JSON	*JavaScript Object Notation*
SGBD	Sistema de Gerenciamento de Banco de Dados
UID	Diagramas de Interação com Usuário (User Interaction Diagrams)
Web	*World Wide Web*
SUT	Sistema em Teste (System Under Test)
SLIM	Framework de Testes de Integração ou Sistema de Teste (SLIM Test System)
TDD	Desenvolvimento Guiado por Testes (Test Driven Development)
FIT	*Framework* de Testes de Integração (Framework for Integrated Testing)
JUnit	*Framework* para Testes de Unidade
Selenium	Navegadores Automatizados (Web Browser Automation)
Java	Linguagem de Programação
FitNesse	*Framework* de Testes de Aceitação
US-UID	Cenários de Usuário através de UID (User Scenarios through User Interaction Diagrams)
TDD	Desenvolvimento Guiado por Testes (Test-Driven Development)
BBD	Desenvolvimento Dirigido por Comportamento (*Behavior-Driven Development*)

SUMÁRIO

1 INTRODUÇÃO

Análogo ao Desenvolvimento Guiado por Testes (*Test-Driven Development* – TDD) (BECK, 2003), o Desenvolvimento Guiado por Testes de Aceitação (*Acceptance Test-Driven Development* – ATDD) consiste no uso de testes automatizados de aceitação (*Automated Acceptance Tests* – AAT) com a restrição adicional de que estes testes são escritos *a priori* da implementação do código da aplicação (HAYES, DEKHTYAR; JANZEN, 2009). A ideia básica do AAT é documentar os requisitos e resultados esperados em um formato que pode ser automaticamente e repetidamente testado (HAUGSET; HANSSEN, 2008). Os AATs representam as expectativas dos clientes (MILLER; COLLINS, 2001) e foram adotados para o desenvolvimento ágil como promessa de promover a comunicação e colaboração entre as partes interessadas (HAUGSET; HANSSEN, 2008; HAYES; DEKHTYAR; JANZEN, 2009).

Ao capturar requisitos de software de usuários, os desenvolvedores ágeis geralmente usam alguma forma de linguagem natural, permitindo que os requisitos sejam escritos por clientes, usuários ou pelos próprios desenvolvedores. Essas partes interessadas formam uma representação humana central dos requisitos acessíveis a ambos engenheiros de requisitos e clientes (KAMALRUDIN, GRUNDY, HOSKING, 2010). No entanto, devido às ambiguidades e complexidades da linguagem natural (KAMALRUDIN et al., 2013), técnicas textuais (VILAIN, 2002) e do processo de captura, os requisitos apresentam muitas vezes inconsistências, redundância, incompletude e omissões. Neste contexto, de acordo com Fitter e Green (1979) linguagens de computador diagramáticas ou esquemáticas ajudam o usuário na representação de requisitos com informações claras e com compreensão do processo. Alguns exemplos são: a endentação do código fonte, diagramas de Venn ou fluxogramas. Assim, linguagens de computador diagramáticas permitem uma melhor verificação, análise e representações estruturadas, idealmente aplicados para melhorar a qualidade de sistemas (KAMALRUDIN, GRUNDY, HOSKING, 2010).

No entanto, no contexto de ATDD, de acordo com a avaliação de Kamalrudin et al. (2013), não há ferramentas para participação dos usuários finais no processo de especificação de testes de aceitação automatizados. Ainda, os formatados de especificações executáveis (tabelas, modelos e comportamentos por meio de exemplos) não são

avaliados como adequados para representação de requisitos por clientes não técnicos (ALVESTAD, 2007).

Para automação de testes, de acordo com Kamalrudin, Grundy e Hosking (2010), os requisitos textuais em linguagem natural são reconfigurados capturando e destacando apenas as interações essenciais e as responsabilidades do sistema. Contudo, Vilain (2002) propõe uma técnica diagramática para representar a interação entre o usuário e o sistema, especialmente em sistemas de informação, onde há intensa troca de informações. Esta técnica diagramática chama-se Diagramas de Interação do Usuário (*User Diagrams Interaction* – UIDs).

Na tentativa de incluir os usuários finais na especificação dos requisitos, esta dissertação propõe os Cenários de Usuários por meio de Diagramas de Interação do Usuário (US-UIDs). Os requisitos especificados por US-UIDs podem ser vistos como os próprios testes de aceitação e podem ser executados automaticamente por uma ferramenta.

Eles são aplicados especialmente no que diz respeito a sistemas de informação onde existe uma intensa troca de informações.

A relevância desta pesquisa se concentra na área de desenvolvimento ágil que se baseia no ATDD, adicionando os US-UIDs como elementos inovadores ao processo de especificação dos testes de aceitação.

1.1 OBJETIVOS

1.1.1 Objetivo geral

O objetivo desta pesquisa é propor uma especialização da técnica UID (US-UID) para criação de testes de aceitação por usuários.

1.1.2 Objetivos específicos

- Avaliar a criação de US-UIDs por usuários;
- Desenvolver um *framework* que suporte a execução dos US-UIDs como testes de aceitação;
- Avaliar a capacidade de rastreabilidade entre os US-UIDs e o código de aplicação.

1.2 ESCOPO

O paradigma desta pesquisa se enquadra na filosofia positivista (GRANJA, 2015). O paradigma positivista possui uma orientação pragmática em que a compreensão deve ser posta em termos de conhecimentos gerais, que depois devem ser colocados em prática. Por isso, é uma abordagem orientada a problemas que se preocupa com a obtenção de soluções práticas. O escopo dessa pesquisa envolve o estudo experimental com situações controladas. O escopo da pesquisa é relativo à utilização da proposta para a criação e a execução de US-UIDs.

Para o escopo da criação, é considerado um experimento envolvendo os US-UIDs do *Jogo do 8*, construído por participantes não técnicos. Neste contexto, são buscadas informações relativas aos fatores de qualidade na especificação de US-UIDs.

No contexto da execução, é considerado um *framework* desenvolvido e um sistema *web*. Neste contexto, são buscadas informações relativas à capacidade de rastreabilidade de erros e falhas do código de aplicação.

1.3 CONTRIBUIÇÕES

As principais contribuições deste trabalho são:

- A utilização de US-UIDs para a criação de testes de aceitação por usuários não técnicos;
- Um *framework* para execução de US-UIDs;
- Um método progressivo e outro regressivo para construção de US-UIDs.

A estrutura desta dissertação está organizada da seguinte forma: o capítulo dois descreve os fundamentos teóricos desta pesquisa. O capítulo três apresenta a revisão da literatura como o estado da arte. O capítulo quatro detalha os US-UIDs. O capítulo cinco descreve as avaliações da proposta. O capítulo seis apresenta um exemplo de uso da proposta. E, por fim, as considerações finais desta pesquisa são apresentadas no capítulo sete.

2 FUNDAMENTOS TEÓRICOS

Neste capítulo são abordados assuntos relacionados à definição de cenários, casos de uso, histórias de usuários, testes de aceitação (ATDD) e conceitos sobre o TDD. Também são apresentadas as definições do FitNesse ou FIT, e dos UIDs, considerando que estes fundamentos serão utilizados na especificação de US-UIDs.

2.1 CENÁRIOS, CASOS DE USO E HISTÓRIAS DE USUÁRIOS

Os casos de uso são usados no início do processo de desenvolvimento de software, para levantar e compreender os principais requisitos do sistema (WAZLAWICK, 2011). De acordo com Sparx Systems (2010), o caso de uso é um elemento para capturar os requisitos de um sistema. Cada elemento de caso de uso representa um objetivo do usuário quando interage com o sistema.

Por outro lado, cenários são histórias personalizadas de ficção com personagens, eventos, produtos e ambientes (PREECE EL AL, 1994). De acordo com Damas et al. (2005), os requisitos são relatados como cenários que capturam tipicamente exemplos de comportamento do sistema por meio de sequências que descrevem as interações e o software para ser feito e seu desenvolvimento.

De acordo Cowan (2015), as histórias de usuários tem um formato específico, concebido para ajudar o autor a ser descritivo e o leitor (desenvolvedor) a tomar ações. Em consulta com os clientes ou proprietários do produto, a equipe de desenvolvimento divide o trabalho a ser feito em incrementos funcionais chamados de histórias de usuários (AGILE ALLIANCE, 2015).

No contexto do ATDD, Crispin e Gregory (2009) afirmam o seguinte:

> [...] a menos que a indústria decida sobre um vocabulário comum, porque acho que as áreas de negócio da empresa não só entenderão como dar exemplos, mas também compreenderão quando eu falar sobre os testes de aceitação que comprovam a intenção da história ou funcionalidade. A equipe entenderá suficientemente o escopo para então iniciar a codificação e os testes.

Ainda no contexto de ATDD, o desenvolvimento dirigido por comportamento (*Behavior Driven Development* – BDD), de acordo com Hüttermann (2011), promove uma abordagem especial para a escrita e testes de aceitação de aplicação, que são diferentes do TDD, embora o BDD também promova escrever testes primeiro. Com BDD, os testes são como histórias de especificação, normalmente no formato (AGILE ALLIANCE, 2015):

- (Dado) algum contexto;
- (Quando) alguma ação é realizada;
- (Então) um particular conjunto observável de consequências que são obtidas;

Para Cowan (2015) as histórias de usuário ligam as pessoas e o cenário do problema e são validadas no mundo real. A Figura 1 apresenta o ciclo de desenvolvimento considerando as histórias de usuário.

Figura 1 – Processo de desenvolvimento considerando as histórias de usuários

Fonte: Disponível em: <https://www.alexandercowan.com/wp-content/uploads/2014/02/Agile-Development-Process-Design-Thinking1.png>. Acesso em: 15 jun. 2015.

Para Hüttermann (2011), esta técnica de histórias orientada para a especificação também usa uma linguagem natural para assegurar a comunicação *cross-funtional* e para compreender os conceitos de negócio.

De acordo com Butler (2015):

- As histórias de usuários são uma breve declaração que identifica o usuário e a sua necessidade;
- Um cenário de usuário expande suas histórias de usuários, incluindo detalhes sobre como um sistema pode ser interpretado e usado;
- Um caso de uso é realmente apenas uma longa lista de passos que um usuário pode tomar para tentar fazer alguma coisa;
- E os fluxos de usuários são uma forma de documentação visual que ilustra como os usuários encontram o conteúdo, bem como a sequência de suas interações.

Neste estudo, um cenário é considerado o resultado esperado do usuário, com as informações de interação com o sistema, no contexto de ATDD.

2.2 TESTES DE ACEITAÇÃO

De acordo com a ISO/IEC/IEEE Standard 24765 (2010), o termo "testes de aceitação" (*Acceptance Test*) refere-se aos critérios definidos pelo cliente ou usuários para determinar se o sistema será aceito. Um teste de aceitação é uma descrição formal do comportamento de um produto de software, geralmente expresso como um exemplo, ou uma cenário de uso (AGILE ALLIANCE, 2015).

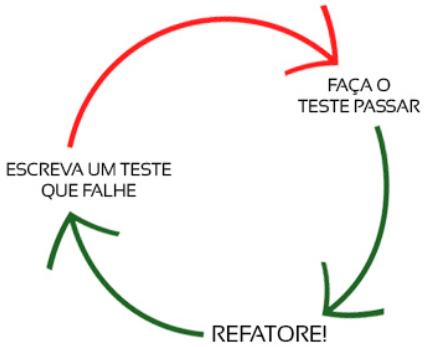
De acordo com Haugset e Hanssen (2008), os testes de aceitação automatizados (AAT – *Automated Acceptance Testing*) foram adicionados aos testes no desenvolvimento de software ágil para comunicação e colaboração. Ainda, de acordo com Kamalrudin et al. (2013), testes de aceitação são conhecidos como testes "funcionais", teste do "cliente" ou "história", e são identificados pelo cliente ou usuário na fase inicial do desenvolvimento.

Para muitas equipes ágeis testes de aceitação são a principal forma de especificação funcional; às vezes a única expressão formal de requisitos de negócios. Em outros casos, eles são meramente complementares a um documento de especificação resultante de uma técnica ágil, como casos de uso ou mais documentos narrativos (AGILE ALLIANCE, 2015).

2.3 TDD

O Desenvolvimento Guiado por Testes (TDD) (BECK, 2003) é uma prática de desenvolvimento de software que consiste em escrever um teste de unidade, executar esse teste e produzir o código de ajuste, caso o teste falhe. Assim que o código está correto é necessário a fatoração ou refatoração para remover códigos duplicados, melhorar os nomes das classes, enfim, deixar o código coeso. Esse processo continua iterativamente, sendo que o teste é automatizado e cada teste é isolado em relação ao outro. A Figura 2 apresenta o fluxo do desenvolvimento iterativo do TDD.

Figura 2 – Desenvolvimento iterativo do TDD



Fonte: Disponível em: <http://tdd.caelum.com.br>. Acesso em: 15 jun. 2015.

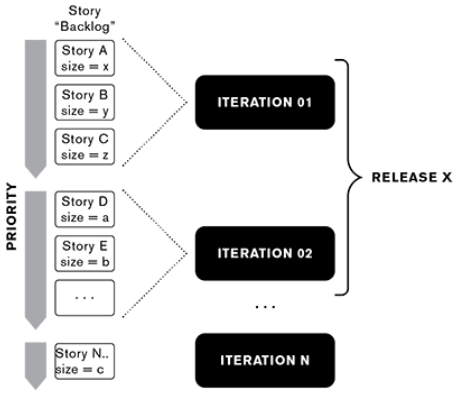
O TDD é uma filosofia, uma cultura, onde as práticas são enfatizadas para o desenvolvimento evolutivo das partes imediatamente necessárias do código de aplicação (BECK, 2003).

Tipicamente, dentro do TDD o desenvolvimento baseado nos testes de aceitação automatizados também é conhecido como ATDD (*Acceptance Test Driven Development*) (GÄRTNER, 2012). Basicamente, os testes de aceitação automatizados ajudam os usuários e o desenvolvedor a documentar os requisitos e resultados desejados em um formato que pode ser automaticamente testado (HAUGSET; HANSSEN, 2008). Os testes automatizados são uma prática da Programação Extrema (XP), sem muita ênfase na distinção entre teste de

unidade e teste de aceitação, e sem notação particular ou ferramenta recomendada (AGILE ALLIANCE, 2015).

O desenvolvimento iterativo incentiva uma melhor gestão de riscos, particularmente em condições de incerteza. Uma iteração tem um conjunto priorizado de histórias como entrada e uma versão do software que funciona como saída. A vantagem desta abordagem é que não são tomadas decisões muito à frente, destacando-se o que realmente sabe-se sobre o que o usuário deseja. O período de tempo de uma interação deve ser de 2-6 semanas (COWAN, 2015). A Figura 3 apresenta o fluxo de iterações ágeis para o desenvolvimento considerando requisitos descritos como histórias de usuários.

Figura 3 – Iterações ágeis



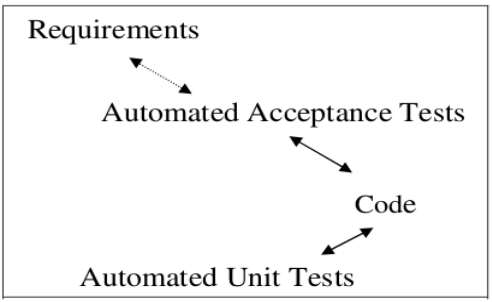
Fonte: Disponível em: < https://www.alexandercowan.com/wp-content/uploads/2014/02/agile-iterations.png>. Acesso em: 15 jun. 2015.

A prática fundamental no TDD é a automação de testes (BECK, 2003). No contexto de ATDD, os testes são derivados de histórias de usuários e implementados em iterações, geralmente como apresenta a Figura 3. Uma das utilidades dos testes automatizados é permitir repetitivamente a execução dos mesmos a fim de verificar o sucesso do código.

A Figura 4, a seguir, apresenta a rastreabilidade dos requisitos ao código. Pode ser observado que os requisitos são especificados por testes de aceitação automatizados que são rastreados até o código de aplicação. O desenvolvimento do código de aplicação depende desta

rastreabilidade para gerenciamento dos requisitos e manutenção do código de aplicação e de teste.

Figura 4 - Rastreabilidade dos requisitos ao código



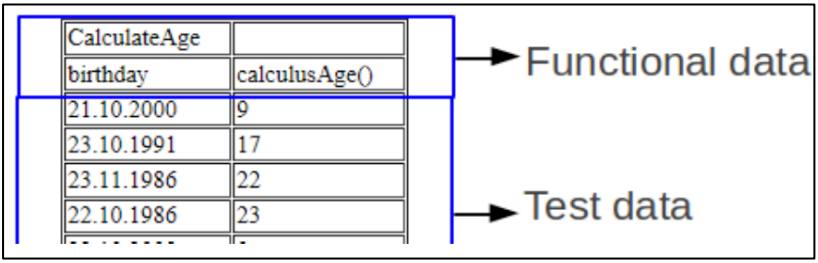
Fonte: (HAYES; DEKHTYAR; JANZEN, 2009).

2.4 FITNESSE OU FIT

FitNesse ou FIT é um *framework* para testes automatizados (HAUGSET; HANSSEN, 2008; HANSSEN; HAUGSET, 2009), sendo utilizado para testes automatizados de regras de negócio e de aceitação, que são expressos por meio de tabelas (HAUGSET; HANSSEN, 2008; HANSSEN; HAUGSET, 2009; BORG; KROPP, 2011; DRUK; KROPP, 2013). As informações das tabelas podem ser divididas em duas categorias (ver Figura 5) (DRUK; KROPP, 2013):

- Dados funcionais - nome de classes, métodos e campos que são ligados à classe de teste;
- Dados do teste - declarações de entradas e saídas esperadas.

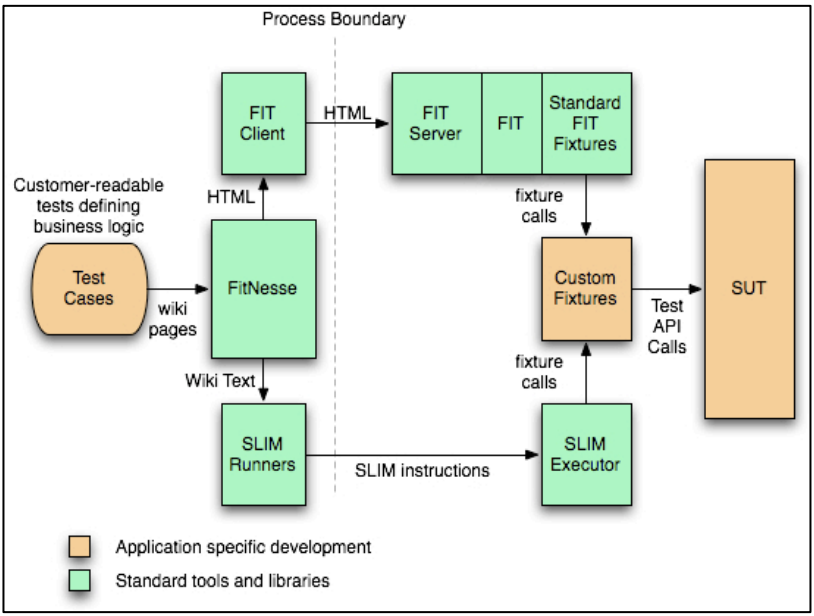
Figura 5 - Uma tabela FIT



Fonte:(DRUK; KROPP, 2013).

Os casos de testes são escritos em páginas *wiki* contendo tabelas. As tabelas podem ser processadas por dois sistemas de teste, FIT ou SLIM. De acordo com Druk e Kropp (2013), os dados funcionais das tabelas são estruturados com os atributos e métodos de uma classe de teste. Esta classe é conhecida como acessório (*Fixture*). O acessório é responsável pela comunicação entre os dados do teste e o sistema em teste (*System Under Test* – SUT). A Figura 6 apresenta a arquitetura do FitNesse.

Figura 6 - Arquitetura do FitNesse



Fonte: Disponível em: <http://www.fitnesse.org/files/fitnesse/images/fitnesse_architecture.jpg>. Acesso em: 15 jun. 2015.

2.5 UID

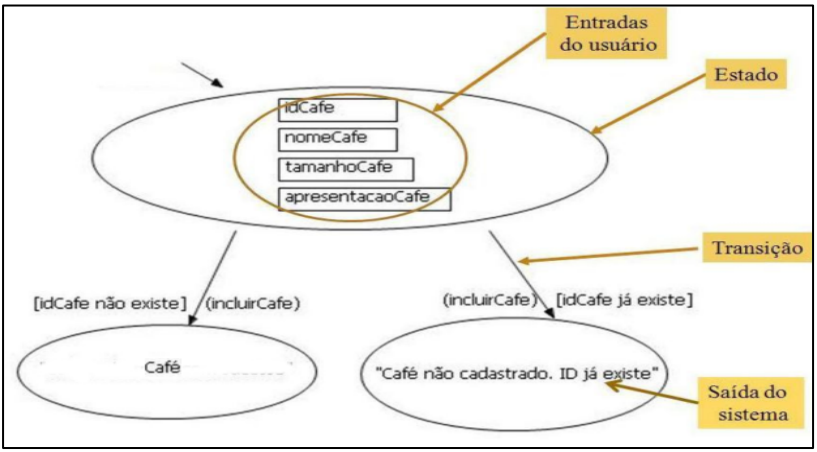
Os Diagramas de Interação do Usuário (UIDs) representam a interação entre o usuário e o sistema, sem considerar aspectos específicos da interface do usuário (VILAIN, 2002; VILAIN SCHWABE e SOUZA, 2000).

A Figura 7 apresenta os elementos de um UID, que são os seguintes:

- Entrada do usuário - é uma caixa que representa um dado fornecido pelo usuário;
- Saída do sistema - é um texto que representa um dado (quando representado sem aspas) ou informação do sistema que é mostrado para o usuário;

- Transição - é a ligação entre os estados de interação, o que troca o foco da interação;
- Estado de interação - é o foco da interação, representado por uma elipse que contém um conjunto de entradas do usuário, saídas do sistema e/ou transições para outros estados.

Figura 7 - Exemplo de UID



Fonte: Adaptado de Giuffra e Vilain (2010).

De acordo com Valderas e Pelechano (2011), os dados que são necessários para apoiar as operações internas do sistema não são descrito nos UIDs. Os UIDs permitem indicar apenas as entidades de dados (por exemplo, nome do café, tamanho do café, café etc.). No entanto, é possível considerar aspectos navegacionais em nível de requisitos (VALDERAS; PELECHANO, 2011).

3 REVISÃO DA LITERATURA

Para realizar a revisão do estado da arte desta proposta, foram consideradas, inicialmente, as seguintes bases de dados: IEEE Xplore, ACM Digital Library e SCOPUS. A pesquisa foi realizada em abril de 2014 e a expressão de busca utilizada foi:

```
(          "tool"
  OR       "tools"
  OR       "MBT"
  OR       "Model-based testing"
  OR       "fitnesse"
  OR       "fit"
)
AND
(
           "automated acceptance test"
  OR       "automatic acceptance test"
  OR       "executable acceptance test"
  OR       "user acceptance test"
  OR       "customer acceptance test"
  OR       "automated acceptance testing"
  OR       "automatic acceptance testing"
  OR       "executable acceptance testing"
  OR       "user acceptance testing"
  OR       "customer acceptance testing"
  OR       "storytests"
)
AND
(
           "requirements"
  OR       "requirement"
  OR       "specification"
  OR       "specifications"
  OR        "story").
```

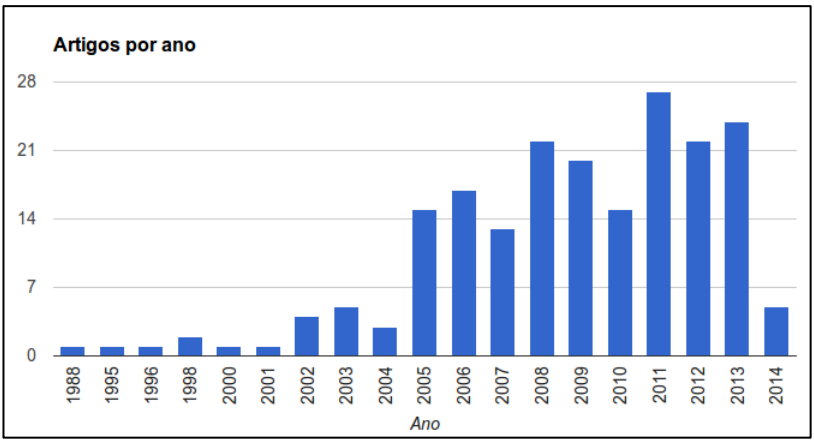
O Quadro 1, a seguir, apresenta o resultado da quantidade de artigos encontrados nas bases de dados.

Quadro 1 - Resultado por base de dados

Base de dados	Endereço	Resultado
IEEE Xplore	<http://ieeexplore.ieee.org>	11
ACM Digital library	<http://dl.acm.org>	89
SCOPUS	<http://www.scopus.com>	111
Artigos		199

O gráfico da Figura 8 apresenta os artigos encontrados por ano de publicação.

Figura 8 - Gráfico dos artigos encontrados por ano de publicação



Várias pesquisas (27 artigos não repetidos) foram descartadas pois estão relacionadas com Engenharia Elétrica, Engenharia de Materiais e testes automatizados de aceitação na etapa de implantação do software. Algumas pesquisas (18 artigos não repetidos) relacionadas com questões sobre modelos de representação de requisitos não foram selecionadas. Muitas destas pesquisas (68 artigos não repetidos) tratam de aspectos voltados para a arquitetura ou usabilidade de ferramentas, aplicações, outras ainda para a questão de custo e tempo. No entanto, foram selecionadas para dois interesses: criação ou especificação de requisitos (Quadro 2), e execução ou automação de requisitos (Quadro 3). As pesquisas do Quadro 2 são muito relacionadas com esta dissertação. As pesquisas do Quadro 3 são menos relacionadas pois

tratam de assuntos da evolução do desenvolvimento e problemas relacionado com a automação de testes.

Quadro 2 - Destaques de pesquisas para especificação de requisitos

Autores	Destaques
Alvestad (2007)	Avaliação das representações de requisitos executáveis.
Kamalrudin et al. (2013)	Avaliação das ferramentas para testes de aceitação automatizados.
Haugset e Hanssen (2008)	Avaliação dos testes de aceitação automatizados para verificar a melhora na comunicação e a colaboração do usuário com o desenvolvedor.
Miller e Collins (2001)	Uma ferramenta baseada em cenários

Quadro 3 - Destaques de pesquisas para automação de testes

Autores	Destaques
Liebel, Alégroth e Feldt (2013)	O estado atual dos testes de aceitação em sistemas com interface gráfica.
Borg e Kropp (2011)	Fatoração em testes de aceitação automatizados.
Druk e Kropp (2013)	Fatoração em testes de aceitação automatizados, especificamente para o FIT.
Greiler et al. (2013)	Aborda problemas de acessórios dos testes identificados durante a evolução do software.
Hayes, Dekhtyar e Janzen (2009)	Rastreabilidade de códigos por testes automatizados.

Os modelos de representação de requisitos para testes de aceitação automatizados encontrados na literatura são: formal, semiformal e informal. O modelo formal consiste em uma linguagem com notações precisas (NAKAGAWA; TAGUCHI; HONIDEN, 2007), em que o usuário precisa ter amplo conhecimento para especificação (KAMALRUDIN, 2009). A vantagem de um modelo formal é que os testes podem ser executados por meio da linguagem que descreve o testes, como por exemplo, na linguagem Java.

O modelo semiformal é baseado no modelo formal com recurso informal (KAMALRUDIN; GRUNDY, 2011).

O modelo informal é baseado em linguagens naturais, histórias e tabelas (KAMALRUDIN; GRUNDY, 2011).

O Quadro 4 apresenta uma avaliação empírica que considera os modelos de especificação em relação à automação de testes e especificação de testes.

Quadro 4 - Relação do formato dos modelos com a automação de testes e especificação de requisitos

Modelo	Automação (por parte do desenvolvedor)	Especificação (por parte do usuário não técnico)
Formal	**Fácil**	**Impossível**
Semi-formal	**Média**	**Difícil**
Informal	**Difícil**	**Fácil**

Os critérios para avaliação apresentado no Quadro 4 são para:

- Automação – indica o esforço do desenvolvedor para converter os requisitos em testes automatizados. O esforço fácil indica que os requisitos são diretamente automatizados. O esforço médio indica que para automatizar os testes são necessários alguns ajustes. O esforço difícil indica que para automação é necessário a conversão ou sincronia com um modelo e perda na derivação do requisito.
- Especificação – indica o esforço ou dificuldade do usuário não técnico para especificar os requisitos. A dificuldade impossível indica que o usuário não técnico não possui capacidade de expressar os requisitos no formato correto. O esforço difícil indica que o usuário poucas vezes conseguirá expressar suas expectativas no formato. O esforço fácil indica que o usuário consegue expressar seus requisitos no formato correto.

3.1 CARACTERÍSTICAS DAS FERRAMENTAS PARA ATDD

As características da ferramenta FIT já foram apresentada na seção 2.4. As características das outras ferramentas apresentadas no Quadro 5 são detalhadas nas próximas subseções, exceto a proposta deste trabalho.

3.1.1 Concordion

Concordion é uma ferramenta *open source* para escrever testes de aceitação automatizados no ambiente de desenvolvimento Java (POPOVIC, 2015). A atividade de especificação consiste em duas partes (KAMALRUDIN et al., 2013):

- Descrição da funcionalidade do sistema em um documento XHTML formal. A Figura 9 apresenta um exemplo de requisito especificado no formato Concordion;
- Implementação de código de acessório (fixture) que implementa uma extensão do padrão JUnit *test case*. A Figura 10 apresenta um exemplo de acessório.

Figura 9 - Formato de especificação Concordion

```
<html xmlns:concordion="http://www.concordion.org/2007/concordion">
    <head>
        <title>System Login</title>
    </head>
    <body>
        <h1>REQ-001 System Login</h1>
        <p>
            The system shall provide system logon function that will be used when
            a user attempts to use the system. The user needs to provide credentials
            in a form of username and password pair.
        </p>
        <div class="example">
            <h3>Examples</h3>
            <p>
                For the purpose of demonstration the system under test contains a
                system login function with username and password hardcoded to
                "johndoe" and "123abc!@#" respectively. All other combinations
                should fail.
            </p>
            <table concordion:execute="#valid=systemLogin(#username, #password)">
                <tr>
                    <th concordion:set="#username">Username</th>
                    <th concordion:set="#password">Password</th>
                    <th concordion:assertEquals="#valid">Sucess</th>
                </tr>
                <tr>
                    <td>john</td>
                    <td>doe123abc!@#</td>
                    <td>no</td>
                </tr>
                <tr>
                    <td>admin</td>
                    <td>admin</td>
                    <td>no</td>
                </tr>
                <tr>
                    <td>johndoe</td>
                    <td>123abc!@#</td>
                    <td>yes</td>
                </tr>
            </table>
        </div>
        <h2>Further Details</h2>
        <ul>
            <li>How to create user and password?</li>
            <li>Username restrictions and validation?</li>
            <li><a href="PasswordValidation.html">
```

Fonte: Disponível em: < https://www.academia.edu/7698408/Concordion>. Acesso em: 15 jun. 2015.

Figura 10 - Um acessório Concordion

```
package exampleapp.spec.login;

import org.concordion.integration.junit3.ConcordionTestCase ;
import exampleapp.Login; // systemunder design

public class PasswordValidationTest extends ConcordionTestCase {

    public boolean isValid(String password) {
        Login login = new Login();
        return login.validatePassword(password);
    }
}
```

Fonte: Disponível em: <https://www.academia.edu/7698408/Concordion>. Acesso em: 15 jun. 2015.

3.1.2 Cucumber

Cucumber é uma ferramenta *open source* para escrever testes de aceitação automatizados, especialmente para Ruby. O Cucumber suporta os requisitos no formato de comportamento (KAMALRUDIN et al., 2013; CUCUMBER, 2015). Os cenários seguem um esqueleto no formato do BDD: dado, quando e então. A Figura 11 apresenta um exemplo do formato de especificação do Cucumber.

Figura 11 - Formato de especificação Cucumber

```
Feature: Addition
  In order to avoid silly mistakes
  As a math idiot
  I want to be told the sum of two numbers

  Scenario: Add two numbers
    Given I have entered 50 into the calculator
    And I have entered 70 into the calculator
    When I press add
    Then the result should be 120 on the screen
```

Fonte: Disponível em: < https://cucumber.io/images/front-page/feature.png>. Acesso em: 15 jun. 2015.

3.1.3 AcceptSoftware

O AcceptSoftware é uma ferramenta para testes de aceitação automatizados que permite o desenvolvimento de software com ATDD simples, mas poderoso (KAMALRUDIN et al., 2013; SAMADHIYA e RANJAN, 2011). A Ferramenta AcceptSoftware estende EasyAccept[1]

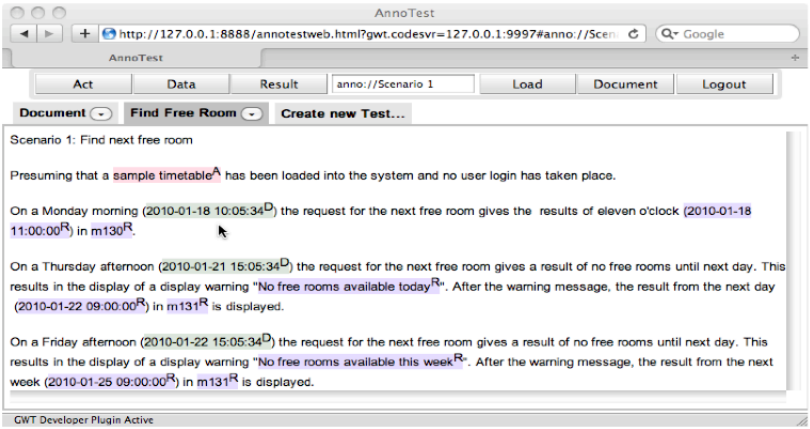
[1] EasyAccept - http://easyaccept.sourceforge.net/

mantendo um histórico de resultado do teste de aceitação. Os testes de aceitação são escritos em arquivos de texto com comandos criados pelo usuário perto de linguagem natural (SAMADHIYA e RANJAN, 2011).

3.1.4 AnnoTestWeb/Run

O AnnoTestWeb/Run é uma ferramenta baseada em navegador, que foi construída usando o Google Web Toolkit e o CouchDB (CONNOLLY, KEENAN e MC CAFFERY, 2010). Ela é destinada para clientes técnicos e permite que eles escrevam testes de aceitação com auxílio de anotações em documentações existentes (KAMALRUDIN et al., 2013). A Figura 12 apresenta um exemplo de requisito e o formato de especificação do AnnoTestWeb/Run.

Figura 12 - Formato de especificação AnnoTestWeb/Run



Fonte: (CONNOLLY, KEENAN e MC CAFFERY, 2010).

3.1.5 UCAT

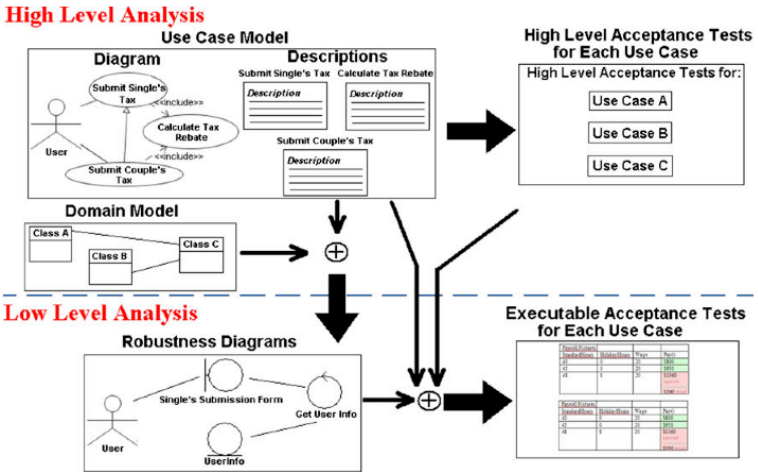
O Use Case Acceptance Tester (UCAT) introduz uma técnica que utiliza o modelo de caso de uso (EL-ATTAR e MILLER, 2010). Esta ferramenta depende da ferramenta FIT/FitNesse (KAMALRUDIN et al., 2013). O processo de criação de testes de aceitação de casos de uso (UCs) consiste em três fases principais (EL-ATTAR e MILLER, 2010):

- Fase 1: Desenvolvimento de testes de aceitação de alto nível (HLATs) para cada UC (*User Case*).

- Fase 2: Realização da análise de robustez para identificar informações de nível de objeto que vai perceber os HLATs criados anteriormente na Fase 1.
- Fase 3: Desenvolvimento de EATs (*executable acceptance tests*) para cada UC usando as informações de nível de objeto identificado na Fase 2.

A Figura 13 apresenta o processo global de desenvolvimento de testes de aceitação de alto nível e executáveis.

Figura 13 - O processo global de desenvolvimento de testes de aceitação de alto nível e executáveis.

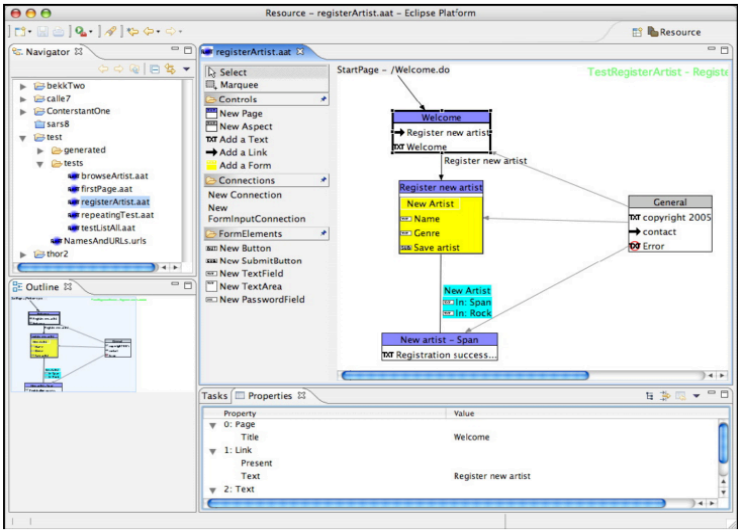


Fonte: (EL-ATTAR e MILLER, 2010).

3.1.6 AutAT

O CubicTest, anteriormente conhecido como AutAT, é um *framework* para testes de aceitação automáticos de aplicações Web (ALVESTAD, 2007). A intenção dos desenvolvedores envolvidos em CubicTest é facilitar a especificação, por usuários não-técnicos, de testes para aplicações Web (ALVESTAD, 2007). O formato de especificação é baseado em elementos das páginas Web e descrições textuais. A Figura 14 apresenta um exemplo de requisito no formato AutAT.

Figura 14 – Formato de requisito AutAT



Fonte: (ALVESTAD, 2007).

3.1.7 RSpec

O RSpec é uma biblioteca Ruby que fornece uma linguagem de domínio específico destinado a BDD (ALVESTAD, 2007). A linguagem Ruby é usada para descrição de comportamentos. A Figura 15 apresenta um exemplo de BDD.

Figura 15 – Um exemplo de BDD

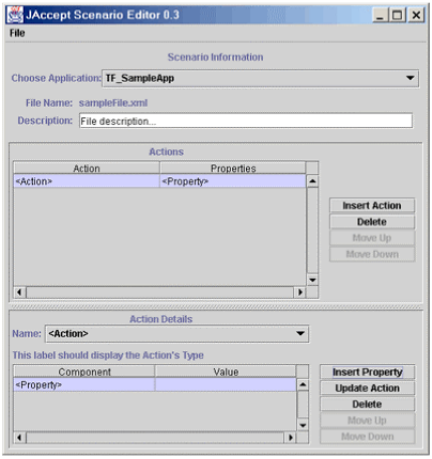
```
describe Account, " when first created" do

  before do
    @account = Account.new
  end

  it "should have a balance of $0" do
    @account.balance.should eql(Money.new(0, :dollars))
  end

  after do
    @account = nil
  end

end
```

Fonte: (ALVESTAD, 2007).

3.1.8 Accept™

O Accept é uma ferramenta para testes de aceitação automatizados, proposta por Miller e Collins (2001). O formato dos testes são cenários descritos por uma lista de ações e propriedades (MILLER e COLLINS, 2001). A Figura 16 apresenta o editor de cenários para o formato Accept.

Figura 16 - Editor de cenários Accept



Fonte: (MILLER e COLLINS, 2001).

3.2 AVALIAÇÃO DAS FERRAMENTAS PARA ATDD

O Quadro 5 apresenta a avaliação de ferramentas para testes de aceitação automatizados. As ferramentas: Concordion, Cucumber, AnnoTestWe b/Run, UCAT, AcceptSoft Ware, são classificadas por Kamalrudin et al. (2013). De acordo com a pesquisa de Alvestad (2007), faltou evidências para aceitar as ferramentas FIT, AutAT, RSpec, como linguagens para especificação de requisitos funcionais. O Accept™ é um editor gráfico de testes de aceitação baseado em cenários (MILLER; COLLINS, 2001).

Quadro 5 - Avaliação das principais ferramentas para testes de aceitação automatizados

Ferramenta	Representação de requisitos			Inclusão do usuário final
	Formal	Semi-Formal	Informal	
Concordion	X	X		
Cucumber	X	X		
AnnoTestWe b/Run	X			
UCAT			X	
AcceptSoft ware	X	X		
FIT			X	
AutAT		X		
RSpec	X			
Accept™		X		
US-UIDs		X		*[1]

*[1]Para especificação do *Jogo do 8* e um módulo de um sistema *web*.
Fonte: Adaptado (KAMALRUDIN et al., 2013).

Das ferramentas avaliadas por Kamalrudin (2013) (Concordion, Concordion, AnnoTestWe b/Run, UCAT, AcceptSoft ware) e Alvestad (2007) (FIT, AutAT, RSpec, Accept™), não há evidências consistentes para inclusão do usuário final na prática de especificação de testes automatizados. De acordo com Liebel, Alégroth e Feldt (2013), os principais problemas com testes de aceitação para sistemas baseados em interface gráfica são as limitações das ferramentas de testes, os altos custos e o envolvimento do cliente nos testes.

De acordo com Park e Maurer (2010), as histórias de teste (*story-test*) são entregues pelos clientes, em vários formatos, sendo mais comum em blocos de anotações ou formato tabular e os desenvolvedores são responsáveis pela implementação dos testes. Greiler et al. (2013) e Druk e Kropp (2013) abordam problemas nos acessórios dos testes (*fixture*) e propõem ferramentas para manutenção e ajuste do código dos testes.

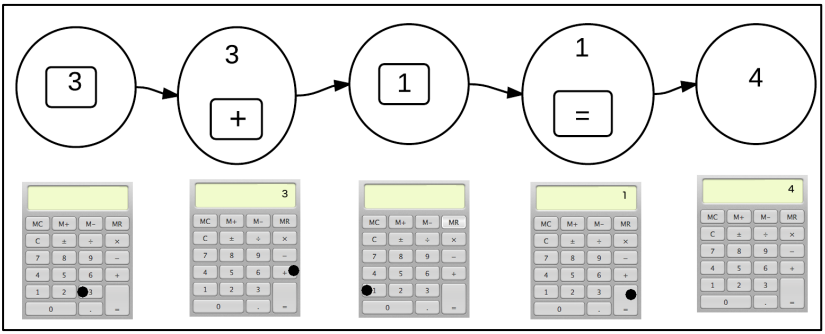
4 CENÁRIOS DE USUÁRIOS POR MEIO DE UIDs

Os cenários de usuários por meio de Diagramas de Interação do Usuário (US-UIDs) são criados e estruturados como uma especialização dos UIDs. Assim, esta seção apresentará a especialização proposta e o *framework* para execução dos US-UIDs.

4.1 US-UIDs

O US-UID é uma especialização da técnica UID onde os tipos de informações dos UIDs são substituídos por valores dos cenários do usuário. A Figura 17 apresenta um exemplo de interação do usuário com uma calculadora e a representação de um US-UID.

Figura 17 - Cenário de usuário de uma calculadora para a função soma



Fonte: Longo e Vilain (2015).

A Figura 17 apresenta a interação do usuário com a calculadora, seguindo o cenário de usuário. O usuário entra com os valores da soma $(3 + 1 =)$ e o sistema apresenta o resultado (4).

O Quadro 6 apresenta o subconjunto dos símbolos dos UIDs que são usados pelos US-UIDs.

Quadro 6 - Símbolos usados pelo US-UID

Símbolo	Utilização
	**Elipse** – é um estado de interação
	Linha direcionada – é uma transição entre os estados e a direção de fluxo.
	**Retângulo** – é uma entrada do usuário, seu valor é representado por um conjunto de caracteres, contido em uma elipse.
Sequência de caracteres	**Valor** – é uma saída do sistema, onde um conjunto de caracteres "e" está contido em uma elipse.

Fonte: Longo e Vilain (2015).

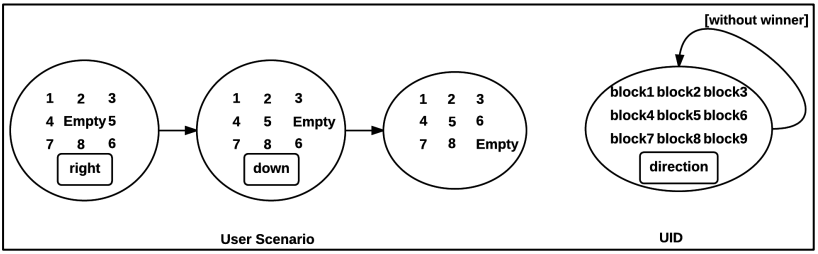
Cada estado de interação (elipse) contém os valores de entrada do usuário e saída do sistema. O fluxo dos estados de interação é representado pelo sentido da transição. O estado de interação inicial é o primeiro estado de interação seguindo o fluxo de direção das setas. O estado final, por sua vez, é o último estado de interação do fluxo.

É importante frisar que apesar dos símbolos usados pelos US-UIDs serem os mesmos dos UIDs, eles são utilizados com diferentes propósitos:

- Em UIDs: para estruturar os nomes das entidades do modelo;
- Em US-UIDs: para estruturar os valores e informações do cenário.

A Figura 18 apresenta um exemplo de um US-UID do *Jogo do 8* e seu respectivo UID.

Figura 18 - US-UID (esquerda) e UID (direita) do *Jogo do 8*

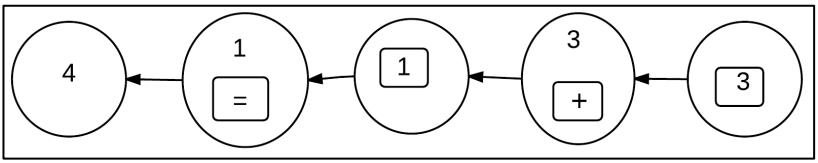


Conceitualmente, seguindo a classificação do formato tabular do FitNesse (DRUK; KROPP, 2013), os US-UIDs podem ser classificados como dados do teste, e os UIDs podem ser classificados como dados funcionais. Contudo, os símbolos usados nos US-UIDs (estados de interação, entradas de usuários, saídas do sistemas e transições) são sincronizados com os respectivos símbolos de um UID. Por exemplo, as entradas de usuário *right* e *down* do cenário de usuário (Figura 18) são associadas à entrada *direction* do UID.

Para construção da especificação dos requisitos como US-UIDs são apresentados os seguintes métodos:

- Progressivo: Indica o resultado esperado apenas no fim do fluxo de construção. Inicialmente são construídos os estados com o objetivo de atingir o resultado esperado. A Figura 17 apresenta o fluxo progressivo de construção da especificação do requisito, ou seja, é construída toda a especificação do método de somar e apenas no fim do fluxo é apresentado o seu resultado.
- Regressivo: Análogo à técnica *Assert First* (BECK, 2003), o método regressivo inicia a construção do US-UID a partir do resultado, e adiciona apenas outros estados para obter o resultado (estado inicial do US-UID). A Figura 8 apresenta o cenário onde o requisito é construído por meio do método regressivo.

Figura 19 - Método regressivo para criar os US-UID da função soma de uma calculadora



Fonte: Longo e Vilain (2015).

Para construir os US-UID com o método regressivo, inicialmente é representado o estado com o resultado esperado (elipse à esquerda, com a saída do sistema), e a partir dele é construído o fluxo regressivo com os outros estados de interações.

4.2 O FRAMEWORK PARA EXECUÇÃO DOS US-UID

Este trabalho propõe um *framework* para automação de testes de aceitação representados por US-UIDs. Esses US-UIDs representam os requisitos do código de aplicação. Deste modo, eles devem ser executados por meio do *framework*, com o objetivo de auxiliar na validação dos requisitos.

O formato dos US-UIDs são executáveis por meio do *framework*, pois os símbolos usados nos US-UIDs são convertidos diretamente em instruções que são executadas em um computador. Uma característica deste *framework* é o suporte a mudanças de cenários ou no código de aplicação. As mudanças nos US-UIDs ou no código de aplicação podem ser avaliadas pela simples execução dos cenários. É importante salientar que os US-UIDs são executados em seu formato original, não sendo convertidos para um código intermediário em alguma linguagem de computador.

4.2.1 Definições de US-UID

Para que os US-UIDs sejam executados pelo *framework*, as informações representadas pelos US-UIDs precisam estar sincronizadas com os dados funcionais. O *framework* considera as seguintes definições na sincronização dos UIDs com os dados funcionais:

$n =$ é *a quantidade de estados de interação de um* $US - UID$.

$a =$ é *a quantidade de entradas de usuário pertencentes a* $State_i$.

$b =$ é *a quantidade de saídas do sistema pertencentes a* $State_i$

$value_i^y =$ é *uma entrada de usuário pertencente a* $State_i$

$value_i^x =$ é *uma saída do sistema pertencente a* $State_i$.

$model_i =$ é *o dado funcional de* $State_i$

$model_i^y =$ é *o dado funcional sincronizado com* $input_i^y$

$model_i^x =$ é *o dado funcional sincronizado com* $output_i^x$

$input_i^y = \{value_i^y, model_i^y\}$

$output_i^x = \{value_i^x, model_i^x\}$

$transition_i = \{source_i, target_i\}$

$target_i =$ é *o estado de interação de destino de uma transição*
$\equiv target_i = State_{i+1}$

$source_i =$ é *o estado de interação de origem de uma transição*

$State_i =$ é *um estado de interação*
$\equiv \{Inputs_i, Outputs_i, transition_i, model_i\}$

$Inputs_i = \{input_i^1, input_i^2, \dots, input_i^a\}, onde: input_i^y \in Inputs_i \ \wedge 0 \leq y \leq a$

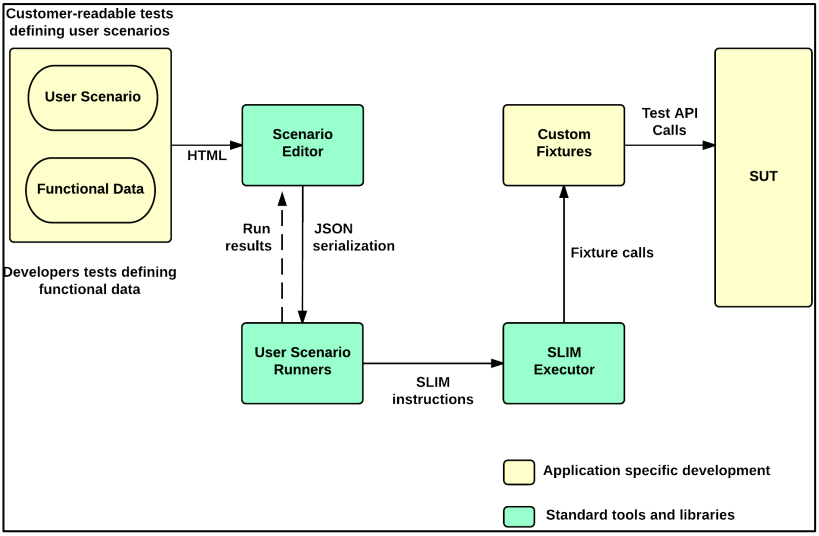
$Outputs_i = \{output_i^1, output_i^2, \dots, output_i^b\}\ onde: output_i^x \in Outputs_i \ \wedge 0 \leq x \leq b$

$US - UID =$ é *um conjunto de estados de interação*
$\equiv \{State_1, State_2, \dots, State_n\}\ onde: State_i \in US - UID \wedge 0 < i \leq n \ \wedge State_1$ é *o estado incial* $\wedge State_n$ é *o estado final*

4.2.2 Arquitetura do Framework

A Figura 20 apresenta a arquitetura do *framework* proposto.

Figura 20 - Arquitetura do *framework* proposto



A arquitetura do *framework* é baseada no FitNesse. Os módulos do *framework* são (Figura 20, verde): editor de cenários; *runner* de cenários; e o executor SLIM. Ainda, os pontos de extensão do *framework* são (Figura 20, amarelo): os US-UIDs e os dados funcionais; os acessórios customizáveis; e o SUT. Os módulos do *framework* e os pontos de extensão são apresentados nas subseções seguintes, onde são mostrados exemplos de US-UID do *Jogo do 8*.

A) **Comparação entre arquitetura do *framework* e a arquitetura do FitNesse.** O objetivo da Figura 20 é apenas mostrar a arquitetura do *framework* proposto. A Figura 20 não apresenta as informações de comparação com o FitNesse para evitar a sobrecarga de informações. Para comparação de similaridade entre o *framework* proposto e o FitNesse devem ser analisadas a Figura 20 e a Figura 6. Deste modo, sem considerar as semelhanças, a seguir são explicas as diferenças dos subsistemas do *framework* proposto e do FitNesse.

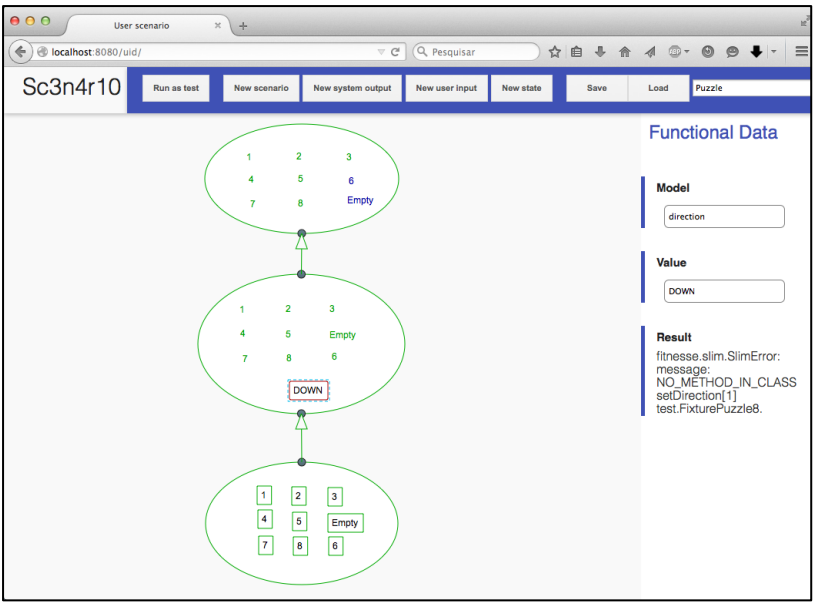
Subsistemas implementados no *framework* proposto. A arquitetura do *framework* proposto é baseada no FitNesse. Assim, os subsistemas ou *frozen spots* implementados são o editor de cenários e o *runner* de cenários, enquanto os pontos extensíveis ou *hotspots* são os US-UIDs e os acessórios customizáveis.

Subsistemas do FitNesse. O executor SLIM é o subsistema do FitNesse reutilizado pelo *framework* proposto. Os pontos extensíveis do FitNesse são: Acessórios customizáveis e o SUT.

4.2.3 Editor de cenários

O editor de cenários permite a criação dos US-UIDs. O acesso ao editor pode ser feito por um navegador. O editor tem suporte apenas ao método regressivo de criação de US-UIDs. Contudo, um dos objetivos principais deste trabalho é a execução de US-UIDs como testes automatizados. A Figura 21 apresenta o editor de cenários.

Figura 21 - Editor de cenários de usuários



Fonte: Elaborado pelo autor.

Os componentes do editor são: os botões de controles (Figura 21, topo); a mesa para estruturar os cenários (Figura 21, centro); e os dados funcionais (Figura 21, direita). Os botões de controle possuem as seguintes funções:

- *Run as test* – executar o US-UID como um teste;
- *New scenario* – criar um novo cenário;
- *New system output* – criar uma saída do sistema;
- *New user input* – criar uma entrada de usuário;
- *New state* – criar um novo estado de interação e a transição ao estado selecionado.

A mesa é um simples quadro de apresentação e edição das estruturas dos US-UIDs. A implementação utiliza a biblioteca Draw2D[2] e permite manipular os símbolos dos cenários de acordo com a estrutura desejada. Os US-UIDs podem ser serializados para JSON, possibilitando o envio para o executor.

Os dados funcionais são compostos pelo *model*, *value* e *result*. O *model* é o nome do acessório ($model_i, model_i^y, model_i^x$) e equivale a um dado funcional em uma tabela FitNesse (Figura 5). O *value* é associado ao dado do teste ($value_i^y, value_i^x$) do símbolo selecionado. O *result* é o estado (erro, falha ou sucesso) da execução do símbolo selecionado.

4.2.4 *Runner* de US-UIDs

O *runner* (corredor) de cenários de usuários é responsável pela conversão de US-UID em instruções SLIM. O *runner* recebe como entrada um arquivo JSON que contém as informações do US-UID que, por sua vez, contém os dados funcionais para execução do cenário. Assim, as instruções são criadas de acordo com o fluxo representado no US-UID.

O *runner* de cenários cria dois tipos de instruções SLIM:

- `fitnesse.slim.instructions.MakeInstruction`: é usada para criar objetos de uma classe. Contudo, a classe deve representar uma *fixture*

[2]Disponível em: <http://www.draw2d.org>. Acesso em: 10 jun. 2015.

correspondente a um estado de interação ($State_i$). Esta instrução é criada para cada um dos estados de interação.

- `fitnesse.slim.instructions.CallInstruction`: é usada para chamar um método. Este método corresponde a uma: saída do sistema ($output_i^x$); entrada de usuário ($input_i^y$); ou transição de estados ($transition_i$).

Após a execução das instruções, os resultados são apresentados pelo editor. Para facilitar a rastreabilidade, cada resultado de instrução é classificado em: sucesso, falha ou erro. O Quadro 7 apresenta a classificação dos resultados das instruções.

Quadro 7 - Classificações dos resultados

Resultado	Cor	Causa
Success	Verde	O código da aplicação foi executado corretamente.
Fail	Azul	O valor esperado é diferente do resultado da execução.
Error	Vermelho	O código da aplicação foi executado incorretamente.

Fonte: Elaborado pelo autor.

Apenas as instruções de saída do sistema podem ser classificadas com resultado de falha. A falha é considerada apenas quando o resultado da instrução de saída do sistema ($output_i^x$) é diferente do valor esperado ($value_i^x$).

4.2.5 Executor SLIM

O executor SLIM pertence ao FitNesse e é o sistema de teste responsável pela invocação das instruções. Para facilitar a implementação das classes de acessórios são feitos os seguintes ajustes nas instruções:

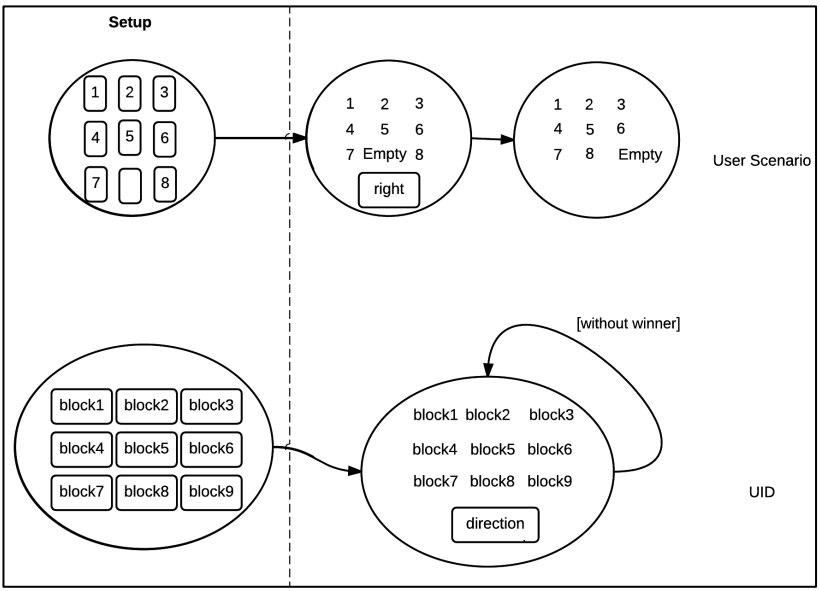
- `MakeInstruction` ($State_i$): é substituído o dado funcional ($model_i$) pela classe adequada do acessório;
- `CallInstruction` ($input_i^y$): é adicionado o prefixo "set" ao nome do método do ($model_i^y$);

- `CallInstruction` ($output_i^x$): é adicionado o prefixo "get" ao nome do método ($model_i^x$);
- `CallInstruction` ($transition_i$): é adicionado o prefixo "to" ao nome do método ($target_i model_i$).

4.2.6 US-UIDs e dados funcionais

Conceitualmente, os US-UIDs são criados pelos usuários e os dados funcionais são definidos pelos desenvolvedores. Assim, no exemplo apresentado nesta Dissertação, são utilizados os US-UIDs do *Jogo do 8*. Em seguida, para cada US-UID do *8-puzzle*, são definidos os dados funcionais do teste pelo editor (Figura 21, direita). Contudo, cada cenário pode necessitar de um estado inicial para execução. Este estado pode ser diferente para cada cenário. Neste caso, é necessário definir um estado adicional para a configuração do US-UID. A Figura 22 apresenta um exemplo de configuração adicional.

Figura 22 - US-UID do jogo *8-puzzle* com configuração adicional e o respectivo UID



Fonte: Elaborado pelo autor.

Os *frameworks* para automação de teste da família xUnit chamam de *fixture setup* o código de configuração para colocar o SUT em condições adequadas para exercício do teste (BECK, 2003, GREILER et al., 2013, LONGO et al., 2015). Similar a esse conceito, o estado adicional de configuração do US-UID (Figura 22, lado esquerdo) contém as precondições necessárias para a execução do US-UID.

4.2.7 Acessórios customizáveis

Os acessórios são responsáveis pela comunicação entre os US-UIDs e o SUT. Os acessórios são construídos com base nos dados funcionais. A Figura 23 apresenta um exemplo de código de um acessório para o US-UID do *Jogo do 8* (Figura 22).

Figura 23 - Exemplo de classe de acessório para troca de informação entre US-UID e o SUT

```
1  package test;
2
3  @Fixture("Puzzle8")
4  public class FixturePuzzle8 {
5
6          private GamePuzzle8 sut = FixtureSetup.getGame();
7
8          // Transition (without winner)
9          public void toPuzzle8() {    }
10
11         // User input
12         public void setDirection(String direction) {
13                 sut.moveEmpty(Directions.forName(direction));
14         }
15
16         // System output
17         public Object getBlock1() {
18                 return sut.getBlock1();
19         }
20                 . . .
21         public Object getBlock9() {
22                 return sut.getBlock9();
23         }
24 }
```

Fonte: Elaborado pelo autor.

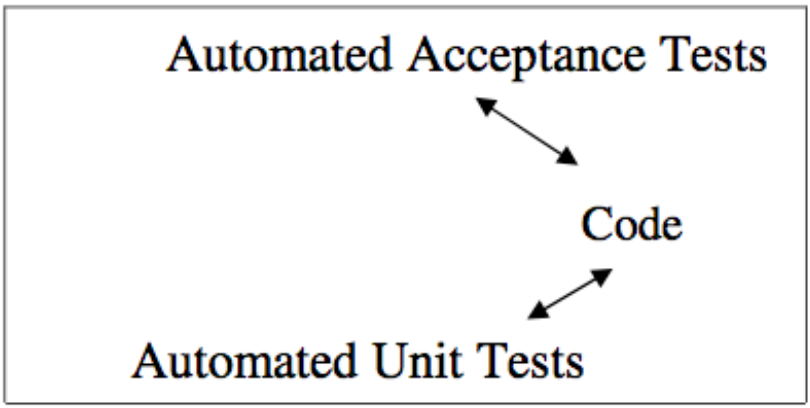
O executor SLIM invoca o SUT por meio da classe do acessório. A classe do acessório (Figura 23) é identificada pela anotação

`@Fixture` (linha 3). Cada estado de interação é associado a uma classe do acessório e cada entrada de usuário é associada a um método "`set`" (linha 12) da classe do acessório. De forma análoga, cada saída do sistema é associada a um método "`get`" (linha 17) e cada transição é associada a um método "`to`" (linha 9). A anotação `@Fixture` foi proposta para identificar as classes do acessório e tornar possível a sincronia com os estados de interação.

4.2.8 SUT

O SUT é executado por meio de invocações das instruções SLIM, associadas às classes de acessórios. Desta forma, as classes de acessórios tornam-se parte do SUT e este pode ser composto por códigos de aplicação, bibliotecas e ferramentas. A Figura 24 apresenta a rastreabilidade da relação entre os testes automatizados e o código de aplicação (HAYES; DEKHTYAR; JANZEN, 2009). A ligação que possibilita esta rastreabilidade é realizada no SUT.

Figura 24 - Rastreabilidade de testes automatizados para código



Fonte: (HAYES; DEKHTYAR; JANZEN, 2009).

A Figura 24 divide os testes automatizados em duas categorias: testes de aceitação e testes de unidade. Contudo, nesta proposta não enfatiza-se a distinção destas categorias.

5 AVALIAÇÕES

Este capítulo apresenta a avaliação dos US-UIDs e a avaliação do *framework* proposto.

5.1 AVALIAÇÃO DOS US-UIDs

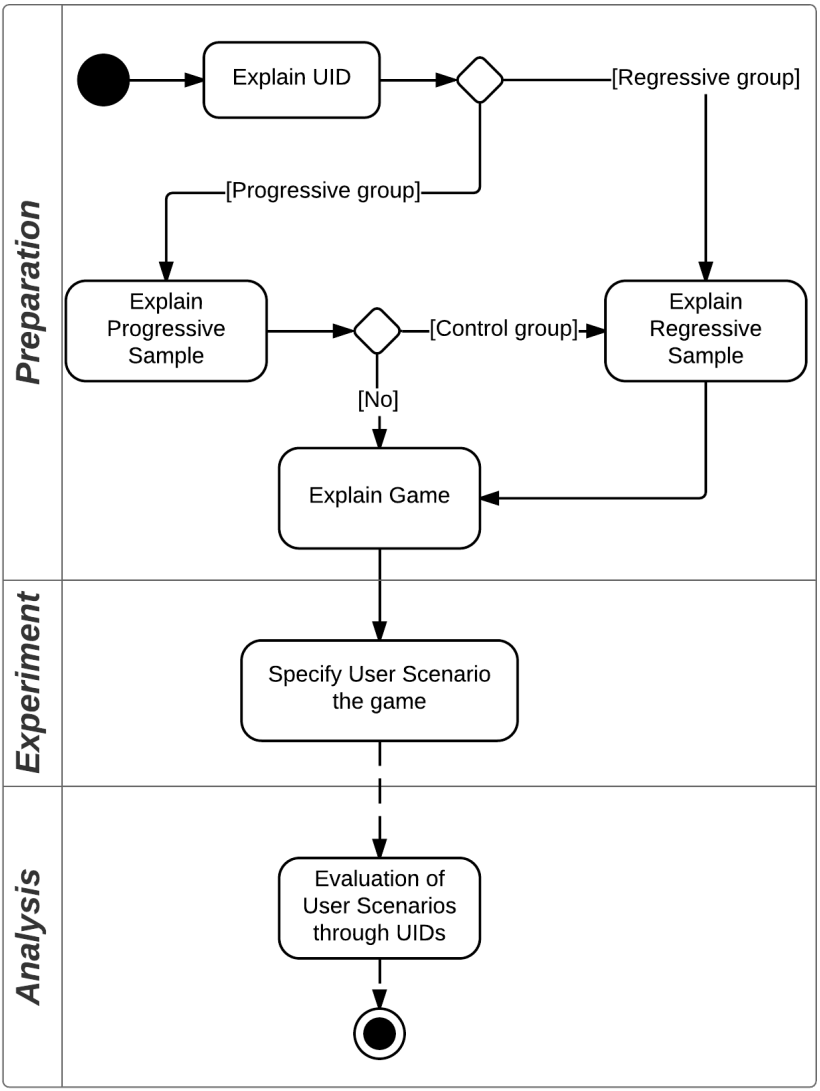
Para avaliar a aplicabilidade e utilidade da proposta em relação à completude e consistência dos requisitos representados por participantes não técnicos conduziu-se um experimento. O experimento é aplicado para construção dos cenários de usuário do *Jogo do 8*, em que é proposto investigar as questões seguintes:

- RQ1: Quais fatores de qualidade (completude e corretude) dos requisitos são representados em cenários de usuário?
- RQ2: Os fatores de qualidade (completude e corretude) são associados aos métodos progressivo ou regressivo?
- RQ3: Qual método proposto facilita a construção dos cenários do usuário?

5.1.1 Metodologia para avaliação

Para avaliação, considerou-se a construção dos cenários de usuário para a especificação dos requisitos sobre o *Jogo do 8*, por participantes não técnicos. Os materiais necessários foram: lápis ou caneta, borracha e papel em branco. A ferramenta de edição dos US-UIDs, apresentada na subseção 4.2.3 não foi utilizada neste experimento pois ela ainda não tinha sido desenvolvida. A Figura 25 apresenta o diagrama com as atividades executadas durante as três etapas da avaliação: preparação, experimento e análise de resultados.

Figura 25 - Diagrama de atividades para avaliação dos cenários de usuário por meio de UID



Fonte: Longo e Vilain (2015).

Durante a preparação da aplicação deste experimento foram registradas algumas informações sobre os participantes. Assim, o espaço amostral foi constituído por 21 participantes, divididos em 3 grupos. O Quadro 8, mostrado na subseção 5.1.4, apresenta mais informações sobre o espaço amostral como as características dos participantes, nível de escolaridade e idade.

5.1.2 Preparação

Para responder as questões e analisar as diferenças entre os métodos, durante a preparação os participantes foram divididos em três grupos: progressivo, regressivo e controle. Na etapa de preparação, os participantes foram treinados por cerca de 15 minutos, de acordo com os grupos. Assim, durante a etapa de preparação do grupo progressivo foram realizadas atividades de explicação sobre: UIDs, método progressivo e o *Jogo do 8*. Para o grupo regressivo foram realizadas as atividades de explicações sobre: UIDs, método progressivo e o *Jogo do 8*. Para o grupo controle, foram realizadas as atividades de explicações de: UIDs, método progressivo, método regressivo e *Jogo do 8*. Assim, cada participante do grupo controle realizou sua própria opção durante o experimento. Durante as explicações, os participantes utilizaram o papel e o lápis para algum treino, quando necessário.

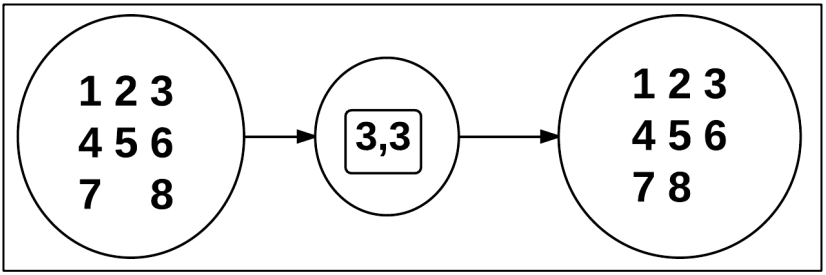
5.1.3 Experimento

Na etapa do experimento, cada participante, individualmente, teve como tarefa especificar o estado final do *Jogo do 8* ou a vitória, considerando alguma interação do usuário para configuração do tabuleiro. A elaboração dos cenários do usuário foi feita utilizando o papel e lápis ou caneta, de acordo com a preferência do participante. Cada participante teve o limite de tempo máximo de 15 minutos para a realização da tarefa do experimento. O tempo gasto por cada participante para realização da tarefa foi coletado durante o experimento.

A) Análise da questão RQ1

A Figura 26 apresenta um cenário de usuário esperado para o *Jogo do 8*.

Figura 26 - US-UID esperado do *Jogo do 8*



Fonte: Longo e Vilain (2015).

As variações dos US-UIDs (criados pelos participantes) como: quantidade de entrada do usuário para realização de movimento das peças (números) do tabuleiro, quantidade de interações e a direção do fluxos das transições, são consideradas adequadas, mesmo sendo diferentes ao US-UID da Figura 26.

A questão RQ1 deve ser respondida por meio da avaliação dos cenários de usuário. A análise considera a avaliação dos fatores de qualidade: completude e corretude.

Completude. Significa que todos os serviços requeridos pelo usuário devem ser definidos (SOMMERVILLE, 2011; LUCIA; QUSEF, 2010). A completude será avaliada atribuindo valores completo/incompleto. É considerado completo o US-UID que apresentar:

- O resultado final ou estado de vitória; e
- Pelo menos uma entrada do usuário para configuração do tabuleiro.

De acordo com Massey et al. (2014), a incompletude é definida como tipo de ambiguidade e ocorre quando uma declaração não fornecer informações suficientes para haver uma única interpretação clara. Considera-se ser incompleta a situação de uso que:

- Não apresentar o estado de vitória;
- Não apresentar a interação de jogar;
- Ou estiver incorreto.

Corretude. É o fator de qualidade que indica se o participante compreende e aplica corretamente a linguagem dos UIDs. São atribuídos valores correto/incorreto para esse fator de qualidade. O valor correto é atribuído aos cenários de usuário onde os símbolos da linguagem são aplicados adequadamente, de acordo com o Quadro 6. O valor incorreto é atribuído ao cenário de usuário que:

- Conter transições cíclicas;
- Conter transições para mais de um estado de interação;
- Conter transições para valores aleatórios; ou
- Não considerar o fluxo de transições.

B) Análise da questão RQ2

A questão RQ2 deve ser respondida por meio da fórmula da seguinte hipótese:

$$H_0 : F_{Progressive} = F_{Regressive}$$

$$\text{e} \qquad (1)$$

$$H_1 : F_{Progressive} \neq F_{Regressive}$$

Onde: $F_{Progressive}$ é a completude e a corretude dos cenários de usuário especificados progressivamente; e

$F_{Regressive}$ é a completude e a corretude dos cenários de usuário especificados regressivamente.

A decisão de aceitar H_0 ou H_1 é feita a partir dos dados coletados do experimento. Ao aceitar H_0, afirma-se que os fatores de qualidade dos requisitos expressos são indiferentes, ou seja, os fatores de qualidades não estão associados ao método. Contudo, ao aceitar a hipótese H_1, afirmamos que os fatores de qualidade são diferentes entre os dois métodos, ou seja, os fatores de qualidade estão associados ao método. O nível de significância do teste estatístico deve ser de pelo menos 5% (α=0.05).

C) Análise da questão QR3

A facilidade ou esforço de construção dos US-UIDs é analisada por meio dos dados de tempo. Assim, deve ser feita a análise de distribuição do tempo gasto pelos participantes de acordo com o método utilizado para comparação.

5.1.4 Resultados

A) O espaço amostral

O experimento foi realizado com 21 participantes. Os participantes não possuem experiência com automação de testes, nem familiaridade relevante com o formato tabular (como FIT) para automação de testes. Durante o experimento foram coletas as características dos participantes de: idade, grau de escolaridade, experiência com computação.

Quadro 8 – Características dos participantes

Participante	Idade	Nível de escolaridade	Experiência com computação	Preparação
C1	**26**	**SC**	**7**	**Progressiva**
C2	**22**	**SI**	**4**	**Progressiva**
C3	**26**	**SC**	**6**	**Progressiva**
C4	**25**	**SI**	**0**	**Progressiva**
C5	**26**	**SC**	**7**	**Progressiva**
C6	**31**	**SI**	**0**	**Progressiva**
C7	**45**	**D**		**Progressiva**
C8	**27**	**SC**	**5**	**Progressiva**
C9	**28**	**SC**	**0**	**Controle**
C10	**25**	**SC**	**0**	**Controle**
C11	**23**	**SC**	**0**	**Controle**
C12	**23**	**SC**	**7**	**Controle**
C13	**27**	**SC**	**11**	**Controle**
C14	**31**	**D**	**13**	**Controle**
C15	**35**	**SI**	**0**	**Controle**
C16	**28**	**SC**	**0**	**Regressiva**
C17	**26**	**SC**	**0**	**Regressiva**
C18	**29**	**SI**		**Regressiva**
C19	**30**	**SC**	**0**	**Regressiva**
C20	**22**	**SC**	**3**	**Regressiva**
C21	**38**	**SC**	**20**	**Regressiva**

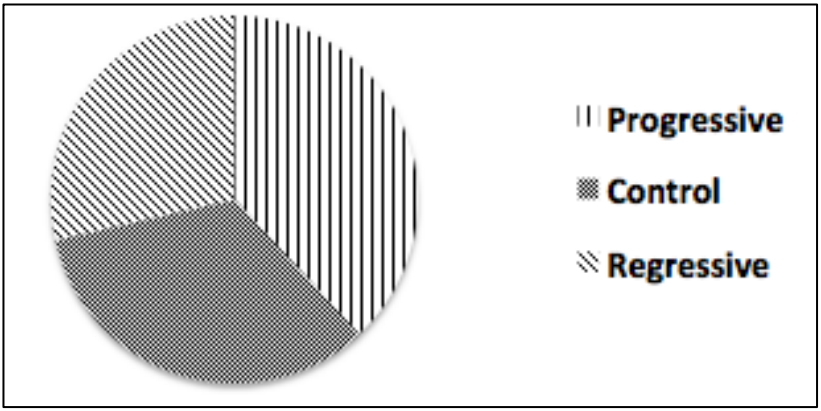
O Quadro 8 apresenta os dados das variáveis:

- Idade – valor em anos relativo ao tempo de vida do participante.
- Nível de escolaridade – refere-se ao nível de educação, D = possui título de doutor; SC = possui título de graduação e/ou está cursando mestrado; SI = está cursando graduação;
- Experiência com computação – refere-se ao valor em anos que realizou ou mantêm atividades relacionadas com computação ou aplicação de computação, essencialmente atividade de programação de computadores.
- Preparação – é relativo aos grupos dos participantes da etapa de preparação descritos na subseção 5.1.1 e cada participante é determinado como progressivo, regressivo ou controle.

B) Configuração da etapa de preparação

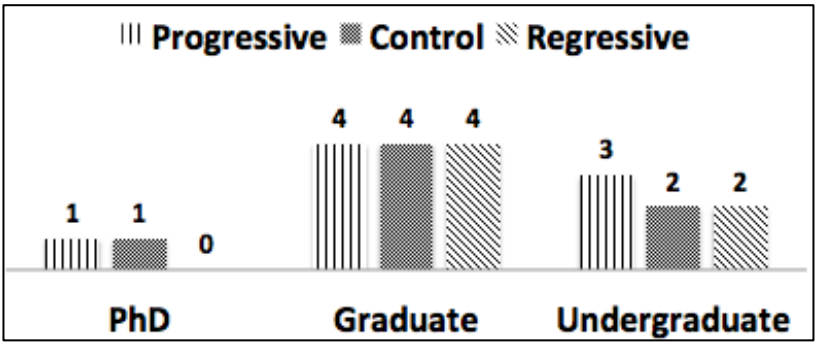
Os 21 participantes foram preparados de acordo com os métodos progressivo e/ou regressivo, resultando em três grupos. O grupo progressivo foi composto por 8 participantes. O grupo regressivo foi composto por 6 participantes. O grupo controle foi composto por 7 participantes. O gráfico da Figura 27 apresenta a distribuição dos participantes por grupos.

Figura 27 – Distribuição de participantes por grupos



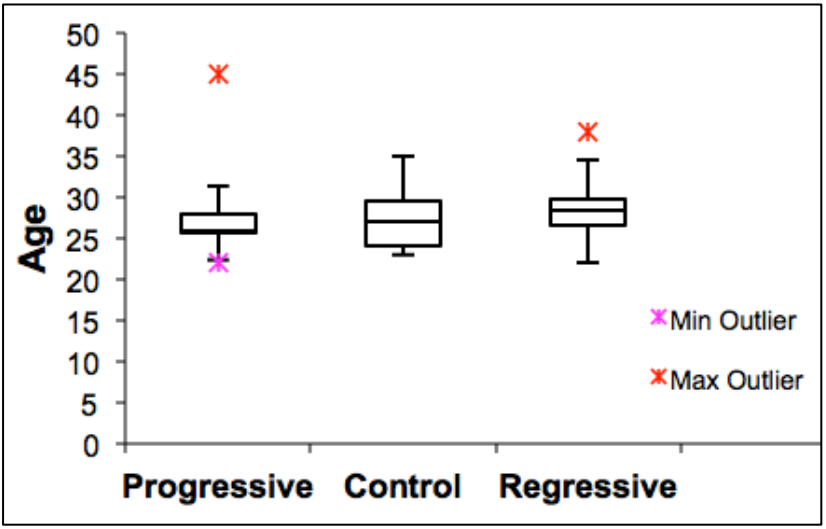
O gráfico na Figura 28 apresenta o nível de educação dos participantes.

Figura 28 -Nível de educação por grupo



As idades dos participantes variam de 22 anos a 45 anos. A Figura 29 apresenta os diagramas de caixa com a variação das idades dos participantes de acordo com o grupo.

Figura 29 - Variação da idade dos participantes por grupos



Fonte: Longo e Vilain (2015).

C) Resultados do experimento

Os cenários de usuário entregues pelos participantes foram inicialmente classificados de acordo com o método progressivo ou regressivo. O método progressivo foi utilizado por dez participantes e o regressivo por doze participantes.

O Quadro 9 apresenta a matriz de correlação entre os métodos usados durante a preparação e os métodos usados por cada participante durante o experimento.

Quadro 9 - Matriz de correspondência entre a preparação e o experimento

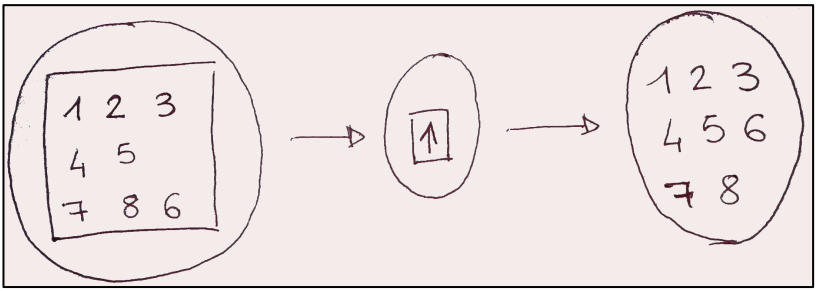
Matrix de correspondência		**Experimento**	
		Progressivo	**Regressivo**
Preparação	Progressivo	8	0
	Regressivo	0	6
	Controle	2	5

Todos os participantes do grupo progressivo usaram o método progressivo durante o experimento. Assim como também todos os participantes do grupo regressivo usaram o método regressivo. Contudo, a escolha do grupo de controle foi:

- 29% método progressivo;
- 71% método regressivo.

Completo e Correto. A avaliação dos fatores de qualidade (QR1) considerou quinze cenários de usuários completos e corretos. Para exemplificar, a Figura 30 apresenta um cenário de usuário que aplicou o método progressivo e que foi avaliado como correto e completo.

Figura 30 - Cenário de usuário completo e correto utilizando o método progressivo

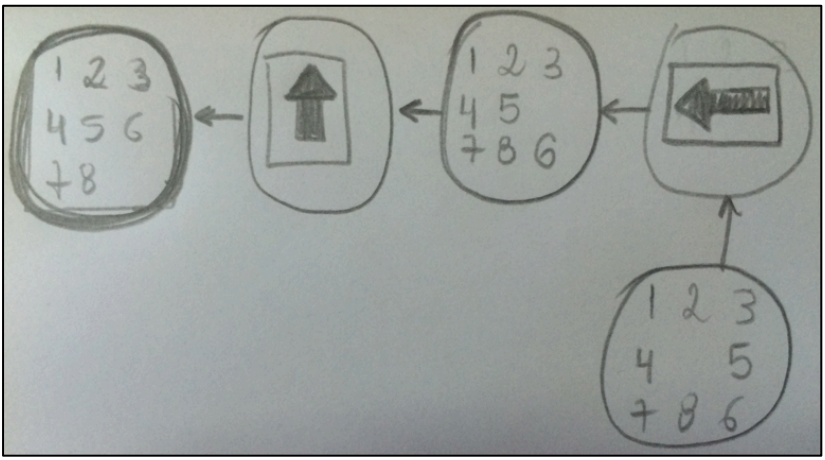


Fonte: Longo e Vilain (2015).

A Figura 31 e a

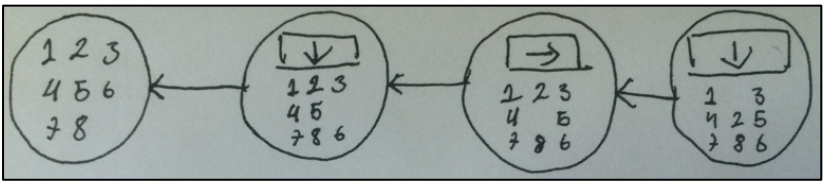
Figura 32 apresentam dois US-UIDs em que se aplicaram o método regressivo e que foram avaliados como corretos e completos.

Figura 31 - US-UID completo e correto utilizando o método regressivo, aplicando movimentos ao número adjacente ao branco



Fonte: Longo e Vilain (2015).

Figura 32- US-UID completo e correto utilizando o método regressivo, aplicando movimentos ao branco

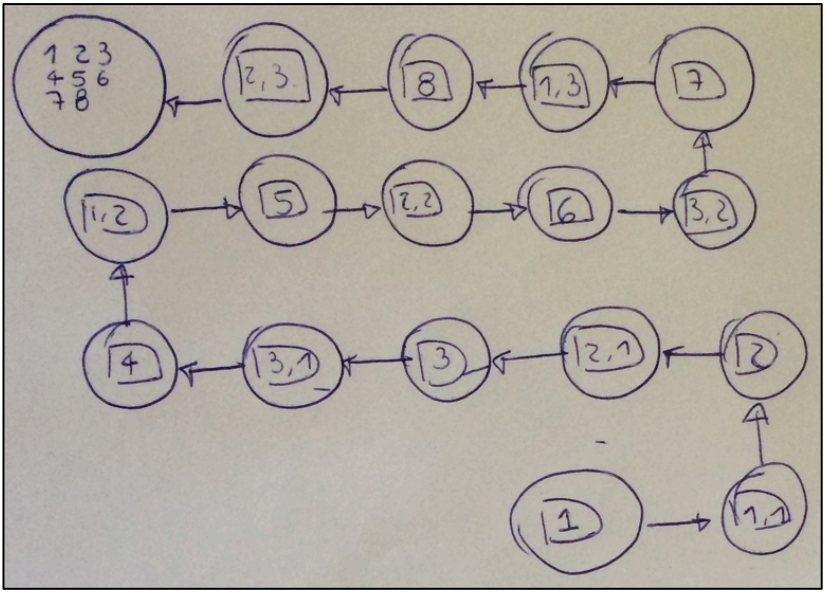


Fonte: Longo e Vilain (2015).

Os US-UIDs da Figura 31 e da Figura 32 apresentam dois tipos de variações para a configuração do tabuleiro do jogo. A Figura 31 apresenta o cenário de usuário que utiliza a entrada do usuário (seta de direção) para mover o número adjacente ao espaço vazio. A Figura 32 apresenta o cenário de usuário que utiliza uma entrada de usuário (seta de direção) que indica o movimento do espaço vazio.

Incompleto e Correto. A avaliação considerou cinco cenários de usuário incompletos e corretos. A Figura 33 apresenta um cenário avaliado como incompleto, em que o participante indicou o preenchimento do tabuleiro, entretanto utilizou corretamente a notação dos UIDs.

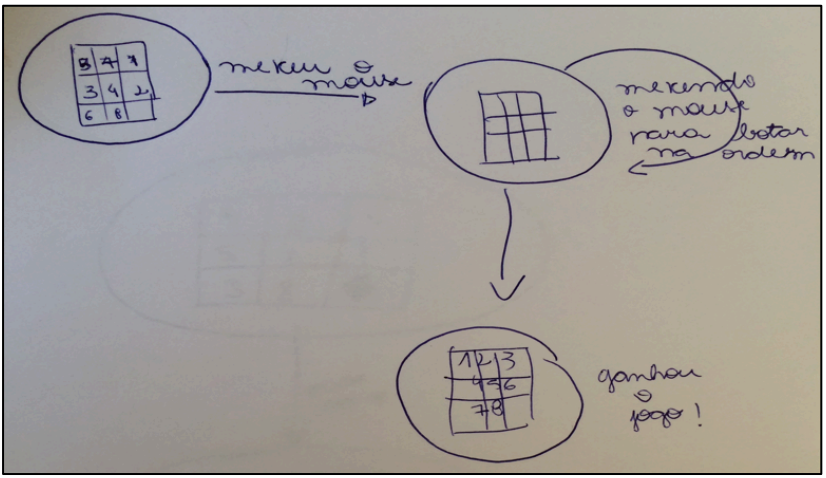
Figura 33 - US-UID incompleto e correto utilizando o método regressivo, não especifica a interação de jogar



Fonte: Longo e Vilain (2015).

Incorreto e incompleto. Em apenas dois casos os participantes usaram os US-UIDs incorretamente. Nestes casos, assumiu-se que o requisito é incompleto pela ausência sintática. A Figura 34 apresenta a tentativa da representação de um US-UID em que o participante omitiu os dados de entrada ou saída e criou ciclos.

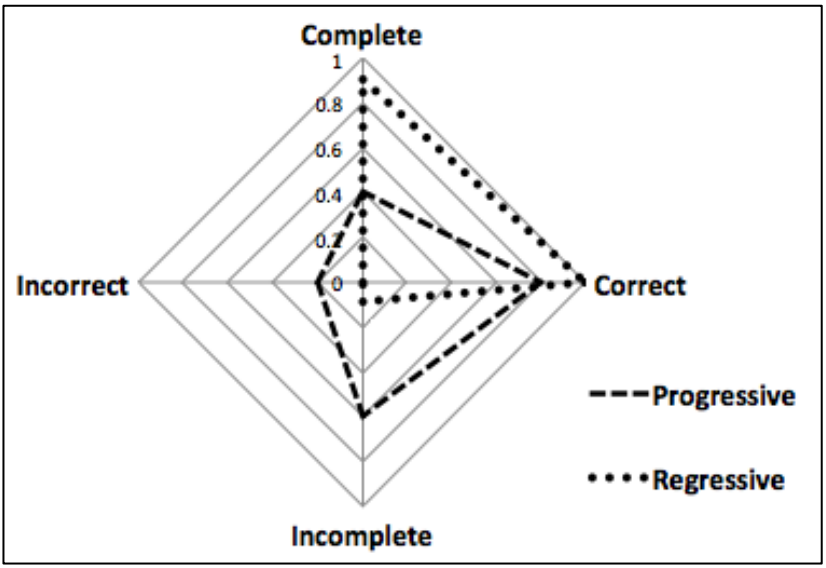
Figura 34 - US-UID incompleto e incorreto utilizando o método progressivo, não é possível compreender o cenário



Fonte: Longo e Vilain (2015).

No resultado da avaliação geral, os participantes conseguiram 67% cenários completos e 90% dos participantes utilizaram corretamente os UIDs. A Figura 35 apresenta o gráfico da distribuição relativa à completude e corretude, seccionada pelo método de construção do cenário.

Figura 35 - Gráfico com a probabilidade de completude e corretude, seccionada por método de construção



Fonte: Longo e Vilain (2015).

O gráfico da Figura 35 apresenta no eixo “x” a distribuição da corretude e no eixo “y” a completude. A área positiva do eixo “x” e “y” representa o melhor fator de qualidade. Pode-se observar que os cenários de usuário especificados por meio do método progressivo apresentam uma probabilidade de que 40% estejam completos e de que 80% estejam corretos. E o método regressivo apresenta 91% de probabilidade de completo e 100% de correto (RQ1).

Para comprovar a diferença dos fatores de qualidade entre os métodos, utilizou-se o teste estatístico de Fisher (LEHMANN; ROMANO, 2006). A Figura 36 apresenta o resultado do teste estatístico de Fisher aplicado sobre os dados do experimento.

Figura 36 - Teste estatístico de Fisher aplicado com a ferramenta estatística R

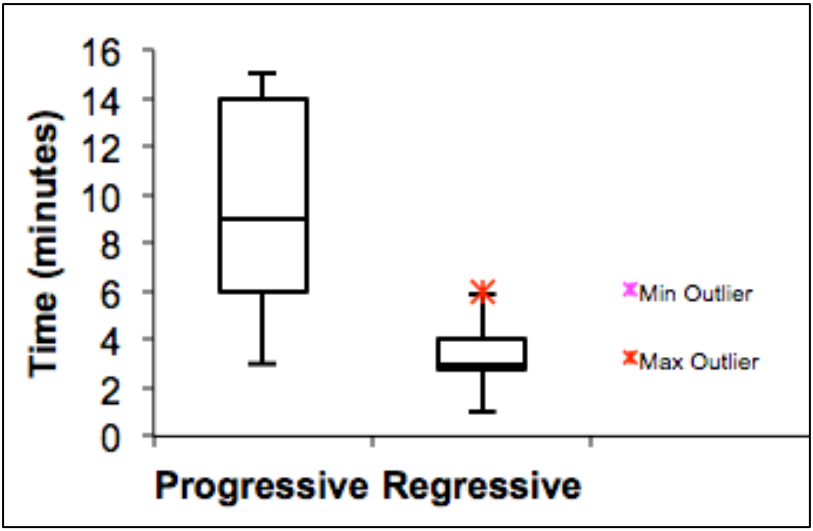
```
> n
            Evalution
Method       Complete and Correct Incomplete and Correct Incomplete and Incorrect
  Progressive                    4                     4                       2
  Regressive                    10                     1                       0
> fisher.test(n)

    Fisher's Exact Test for Count Data

data:  n
p-value = 0.04928
alternative hypothesis: two.sided
```

Fonte: Longo e Vilain (2015).

De acordo com o teste estatístico de Fisher, o p-valor é 0.04928. Entretanto, o nível de confiança do teste deve ser (α=0.05). Assim, $p - value < \alpha$, então, assume-se a hipótese alternativa (H_1), onde conclui-se que os fatores de qualidade são diferentes para os métodos (RQ2).

A Figura 37 apresenta o diagrama de caixa com a distribuição do tempo gasto, seccionado pelo método de construção.

Figura 37 - Distribuição do tempo gasto por método de construção



Fonte: Longo e Vilain (2015).

O tempo gasto pelo grupo progressivo está compreendido entre 3 e 15 minutos e pelo grupo regressivo entre: 1 e 6 minutos. O método regressivo possui um valor discrepante. O valor discrepante (6 minutos) fora da distribuição de tempo gasto pelo grupo indica que um participante pode ter encontrado mais dificuldades para especificar e elaborar o cenário de usuário. As medianas de tempo gasto pelos grupos progressivo e regressivo são de 9 minutos e 3 minutos, respectivamente. Assim, pela análise da distribuição do tempo, o método regressivo, implica em menos esforço (RQ3).

5.1.5 Ameaças da validade

Enquanto o *Jogo do 8* é simples e efetivo para o experimento proposto, abstraído do cenário do jogo, o uso dos US-UIDs para o projeto de requisitos não é fácil. O estudo de caso da pesquisa é incompleto, sem uma discussão de consensos que podem ameaçar a validade dos resultados. A validade interna refere-se às inferências feitas com base nos dados experimentais (YIN, 2003). O objetivo é simplesmente determinar se e como os participantes criaram US-UIDs.

O estudo de caso leva a duas importantes considerações: o conhecimento do caso estudado e o processo de criação dos US-UIDs. O conhecimento do caso estudado, pelos participantes, refere-se ao domínio sobre o *Jogo do 8*, considerando a lógica e regras. Este conhecimento é comum para os participantes. O processo de criação de US-UIDs é diferente do processo e reprodução (cópia) de US-UIDs. Um exemplo de reprodução é copiar o US-UID da calculadora (Figura 17) e simplesmente mudar os valores. O caso estudado evita o processo de reprodução dos participantes, considerando apenas o processo de criação de US-UIDs. O domínio do problema também permitiu a avaliação do esforço sem interrupção por fadiga.

Construir validade refere-se ao uso adequado de métricas de avaliação e medidas (YIN, 2003). Evita-se, especificamente, a utilização de medidas absolutas de completude e corretude, conforme os termos como expressos e aceitos pela padronização (IEEE, 1998). Para calcular as medidas estatísticas, foi utilizada a estatística aceita (*Fisher's Exact Test*) e escrupulosamente seguidas as práticas recomendadas em aplicá-las.

A validade externa refere-se à capacidade de generalizar os resultados para outros domínios (YIN, 2003). A validade externa da pesquisa contém duas ameaças: o domínio do problema estudado e o

domínio da população de participantes. O domínio do problema (*Jogo do 8*) contém interações do usuário com o sistema.

Infelizmente, o estudo utilizou uma pequena população de participantes, em vez de uma grande população de participantes. No entanto, a população de participantes foi composta por diferentes níveis de ensino e áreas de negócios, engenharia e ciências relacionadas.

A confiabilidade refere-se à capacidade de outros investigadores replicarem a metodologia (MASSEY et al., 2014). A proposta e a técnica de avaliação foram detalhadas, bem como os resultados. Considera-se importante que outros pesquisadores consigam reproduzir este estudo.

5.2 AVALIAÇÃO DO FRAMEWORK

A avaliação do *framework* proposto consiste na análise da sua capacidade em rastrear falhas, erros e sucesso nos testes automatizados. Para avaliação foi utilizado um módulo de um sistema *web* desenvolvido com 232 casos de testes automatizados com JUnit[3].

O sistema *web* foi desenvolvido para avaliar e acompanhar os cursos de rede e-Tec Brasil (CISLAGHI, 2014). O sistema *web* foi desenvolvido por meio de Java para as tecnologias da *web*, bem como com o SGBD PostgreSql. A equipe de desenvolvimento e os usuários utilizaram os US-UIDs para comunicação e colaboração durante o desenvolvimento de alguns módulos do sistema *web*. Os US-UIDs foram desenhados em papel e, posteriormente, foram manualmente implementados como casos de teste para o JUnit. Alguns casos de testes utilizam o Selenium[4] pra troca de informações com o navegador.

Para o desenvolvimento do módulo utilizado no experimento, foram criados 4 US-UIDs. O Quadro 10 apresenta as características dos 4 US-UIDs. A implementação manual dos cenários resultou em 232 casos de testes automatizados com JUnit. Enquanto a utilização do *framework* proposto resultou em 231 casos de teste gerados a partir dos 4 US-UIDs utilizados no experimento. A proposta considera que cada entrada de usuário, saída do sistema, estado de interação e transição correspondem a um caso de teste.

[3] Disponível em: <http://junit.org/>. Acesso em: 15 jun. 2015.
[4] Disponível em: <http://www.seleniumhq.org/>. Acesso em: 15 jun. 2015.

Quadro 10 - Características dos cenários utilizados no experimento

US-UID	Estado de interação	Entrada de usuário	Saída do sistema	Transição
A	3	4	25	2
B	5	5	37	4
C	7	6	48	6
D	8	7	57	7

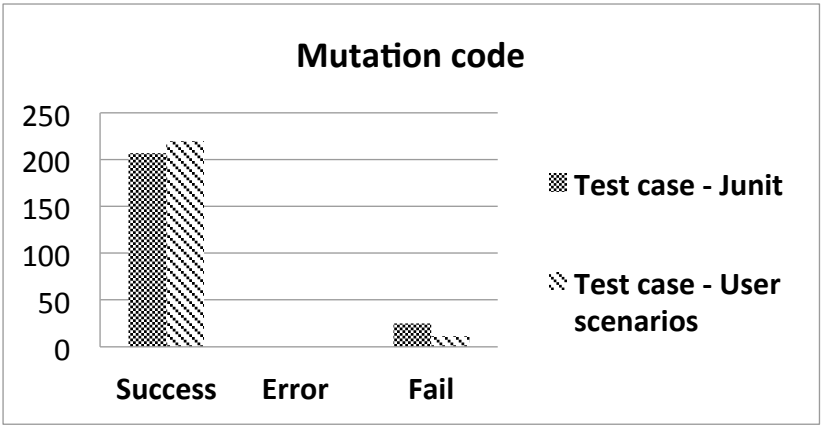
Fonte: Elaborado pelo autor.

A avaliação do experimento consiste na comparação dos 232 casos de teste implementados manualmente com os 231 casos de teste gerados a partir do *framework* proposto. Este experimento não investigou se existe relação na quantidade de casos de teste.

Inicialmente, o experimento consistiu na execução dos casos de teste. Em ambas as abordagens, todos os casos de teste resultaram em sucesso. Após isso, foram realizadas modificações no código da aplicação, o que possibilitou gerar dois experimentos: mutação de código (PAPADAKIS; MALEVRIS; KALLIA, 2010; ARCURI; IQBAL; BRIAND, 2010; MEINKE; NIU, 2010); e falta de código (THONGTANUNAM, 2015).

O experimento de mutação de código foi realizado pela alteração manual do valor de uma lista do código da aplicação. A Figura 38 apresenta o resultado do experimento de mutação de código.

Figura 38 - Resultado da capacidade de rastreabilidade de erros, falhas e sucesso para mutação de código (os resultados são divididos de acordo com casos de testes do JUnit e casos de testes dos US-UIDs)

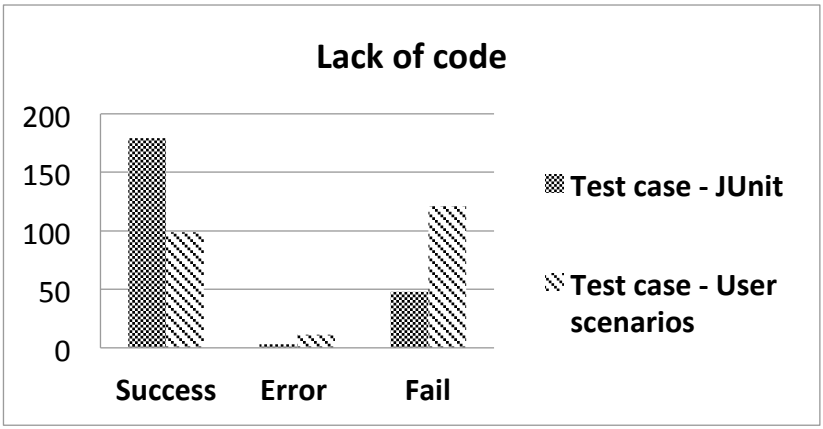


Fonte: Elaborado pelo autor.

O experimento de mutação de código com JUnit resultou em 207 sucessos e 25 falhas, enquanto o experimento com a proposta resultou em 220 sucessos e 11 falhas. Ambos não detectaram erros no código.

O experimento de falta de código foi realizado por meio da remoção de uma classe do código da aplicação. A Figura 39 apresenta o resultado do experimento de falta de código.

Figura 39 - Resultado da capacidade de rastreabilidade de erros, falhas e sucesso para falta de código (os resultados são divididos de acordo com casos de testes do JUnit e casos de testes dos US-UID)



Fonte: Elaborado pelo autor.

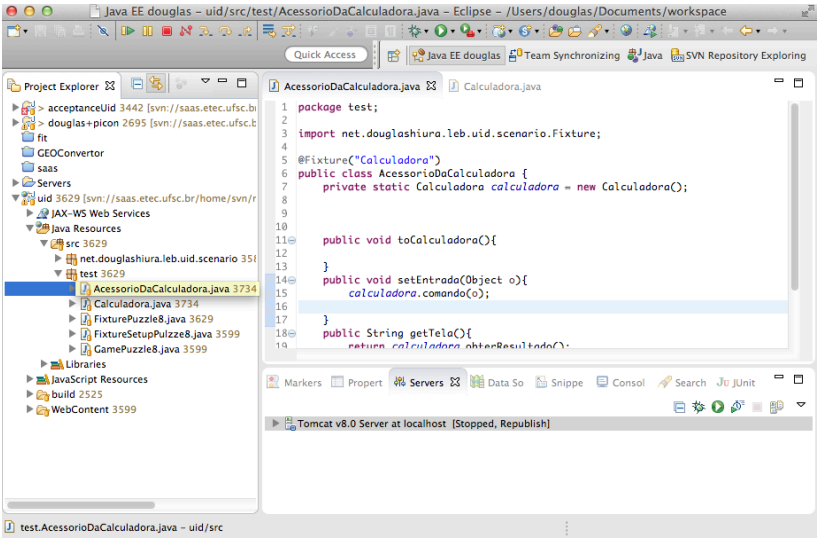
O experimento de falta de código com JUnit resultou em 179 sucessos, 48 falhas e 3 erros, enquanto o experimento com a proposta resultou em 99 sucessos e 121 falhas e 11 erros. Contudo, o *framework* JUnit deixou de identificar 2 dos 232 casos de teste implementados manualmente. Isso ocorreu devido a uma dependência entre os 2 casos de teste e a classe removida. O objetivo deste experimento é apenas verificar se os US-UID são automatizados como testes. Assim, com o experimento foi possível comparar que a rastreabilidade é compatível com a rastreabilidade alcançada com o uso do *framework* JUnit, o que significa um resultado positivo. A principal vantagem de automatizar os US-UIDs com o *framework* desenvolvido é poder executá-los repetidas vezes para verificar a aplicação.

6 EXEMPLO DE USO

Este capítulo apresenta um exemplo de uso que considera a automação de teste com US-UIDs e o desenvolvimento parcial de uma calculadora. As próximas seções apresentam seis passos iterativos para o desenvolvimento, baseados na função soma da uma calculadora.

Para o desenvolvimento do exemplo de uso da calculadora são utilizados o *framework* proposto, a plataforma de desenvolvimento Eclipse[5] e a linguagem de programação Java. Ainda, os US-UID são criados e executados pelo navegador Firefox[6]. A Figura 40 apresenta o ambiente de desenvolvimento da calculadora.

Figura 40 – Ambiente de desenvolvimento da calculadora



6.1 CRIAÇÃO DO US-UID PARA OPERAÇÃO DE SOMA

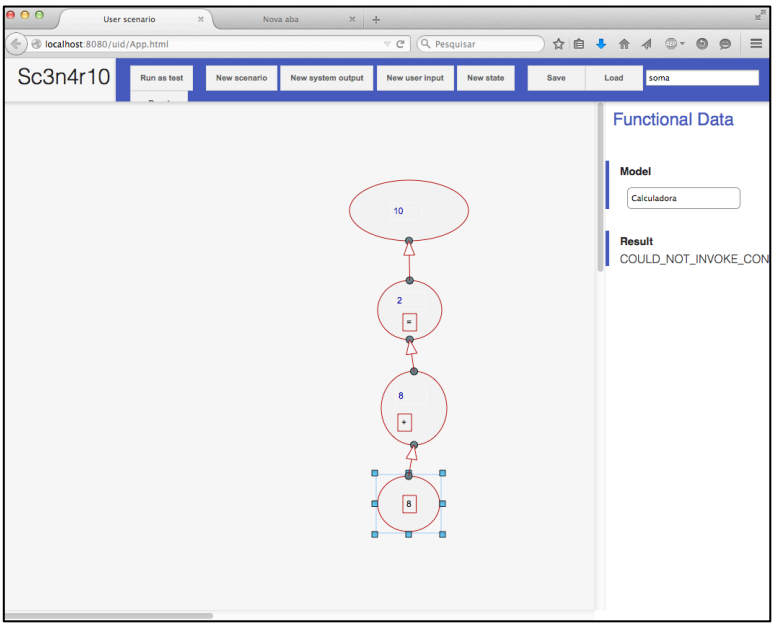
O primeiro passo do desenvolvimento é criar o US-UID, utilizando o editor de US-UIDs, para a operação de soma e definir o modelo. O modelo da calculadora é composto pelos seguintes dados funcionais:

[5] Eclipse - https://eclipse.org/
[6] Firefox - https://www.mozilla.org

- Tela - para saídas do sistema.
- Botão - para entrada do usuário feitas por operações ou números.

Assim, mesmo sem ter implementado o código da aplicação, já é possível executar o US-UID. A Figura 41 apresenta o primeiro US-UID para implementação da calculadora. Como o US-UID apresentado está incompleto, ou seja, falta a entrada do usuário com o valor 2, a sua execução falha e os estados de interação são mostrados em vermelho.

Figura 41 - US-UID para função soma de uma calculadora

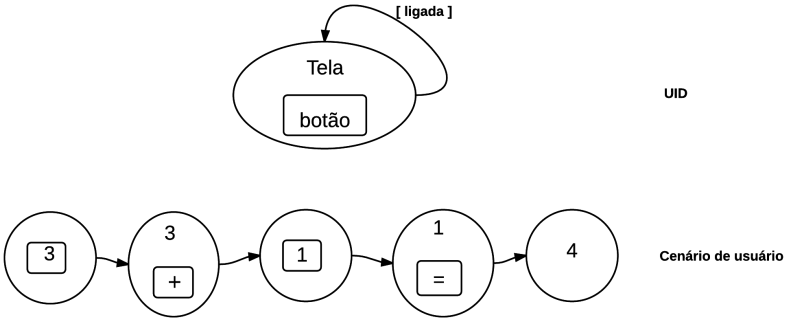


6.2 IMPLEMENTAÇÃO DO ACESSÓRIO

De acordo com os dados funcionais definidos no US-UID, é implementado o acessório. A Figura 42 apresenta o UID para sincronização do modelo funcional com a calculadora. Todos os estados de interação do US-UID são sincronizados com um único estado de interação do UID da calculadora. Este estado chama-se calculadora. As entradas do US-UID são sincronizados com a entrada de usuário que

chama-se de *botão* no UID. As saídas do US-UID são sincronizadas com a saída do sistema chamada de *tela* no UID.

Figura 42 - Sincronização dos dados funcionais com o US-UID da calculadora



A Figura 43 apresenta o código do acessório da calculadora. O código do acessório deve ser implementado e incluído ao SUT. O programador é o responsável por conceber o código do acessório e, então, deve seguir o resultado da execução do US-UID.

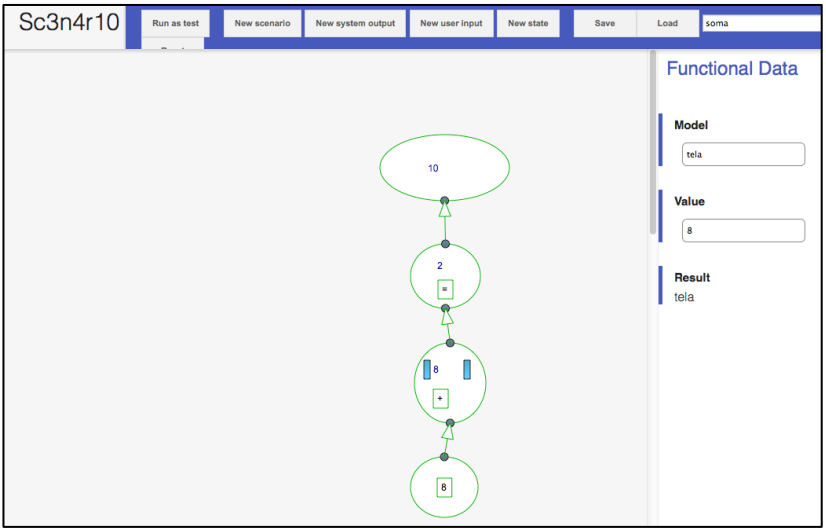
Figura 43 - Acessório para implementação da calculadora

```
package test;

import net.douglashiura.leb.uid.scenario.Fixture;

@Fixture("Calculadora")
public class AcessorioDaCalculadora {

	public void toCalculadora(){}

	public void setEntrada(Object o){}

	public String getTela(){
		return "tela";
	}
}
```

6.3 EXECUÇÃO DO US-UIDS PARA SINCRONIZAÇÃO COM O ACESSÓRIO

Depois de implementado o acessório, é feita a execução para verificar a sincronização do US-UID com o acessório. A Figura 44 apresenta o resultado da execução do US-UID sem erros. Mas é possível rastrear as falhas, ou seja, as saídas do sistemas destacadas de azul.

Figura 44 - Execução do US-UID da calculadora com falhas



6.4 IMPLEMENTAÇÃO DO CÓDIGO DA APLICAÇÃO PARA SUCESSO DA EXECUÇÃO DO US-UID

Para implementação do código de aplicação são recomendados os testes de unidade. Depois da implementação dos testes de unidade e do código da aplicação, é necessário apenas ligar o acessório com o código de aplicação. A Figura 45 apresenta o código do acessório ligado ao código de aplicação. O código de aplicação é criado para satisfazer a execução do teste e sua invocação é feita pelo acessório. Por exemplo, a implementação de uma calculadora pode concentrar todas as entradas em um método chamado *comando*. Então, o acessório para calculadora deve invocar para todas as entradas do sistema o método comando (calculadora.comando(o)).

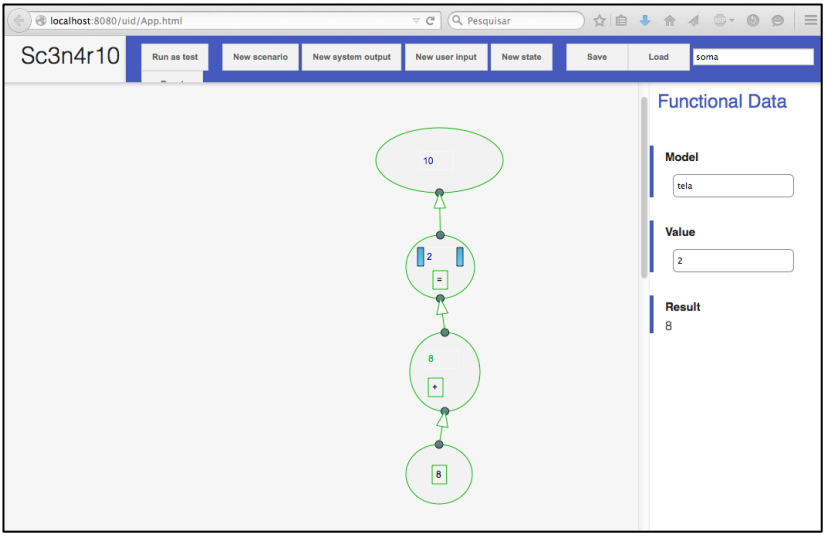
Figura 45 - Acessório ligando ao código de aplicação

```
package test;

import net.douglashiura.leb.uid.scenario.Fixture;

@Fixture("Calculadora")
public class AcessorioDaCalculadora {
	private static Calculadora calculadora = new
Calculadora();

	public void toCalculadora(){

	}
	public void setEntrada(Object o){
		calculadora.comando(o);

	}
	public String getTela(){
		return calculadora.obterResultado();
	}
}
```

6.5 EXECUÇÃO DO US-UID PARA VALIDAÇÃO

Esse passo é para identificar faltas no US-UID, ou seja, identificar se o US-UID está incompleto. Neste caso, o US-UID está incompleto pois falta a entrada do usuário com o valor 2 (Figura 46, saída do sistema destacada em azul). A Figura 46 apresenta execução do US-UID incompleto.

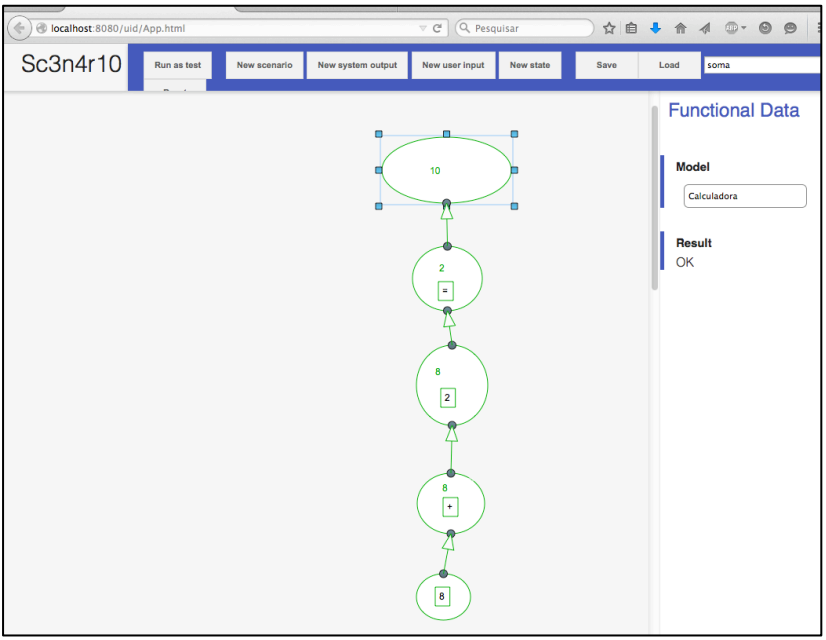
Figura 46 - US-UID incompleto da função soma da calculadora



6.6 AJUSTE E EXECUÇÃO DO US-UID

O ajuste é feito pela edição do US-UID e adição de um novo estado de interação com o valor de entrada 2. A Figura 47 apresenta o US-UID completo para função de soma da calculadora.

Figura 47 - US-UID para função soma adequado ao código de aplicação (Sucesso)



Como pode ser observado na Figura 47, após a execução do US-UID, todos os seus elementos estão verde, indicando que o código cumpre os requisitos do US-UID, ou seja, foram especificados os dados funcionais dos US-UIDs que são sincronizados com os acessórios, e os acessórios invocando adequadamente o código da aplicação.

Os passos apresentados anteriormente (passos 1 a 6) podem ser executados iterativamente, adicionando outros US-UIDs para completar a implementação da calculadora.

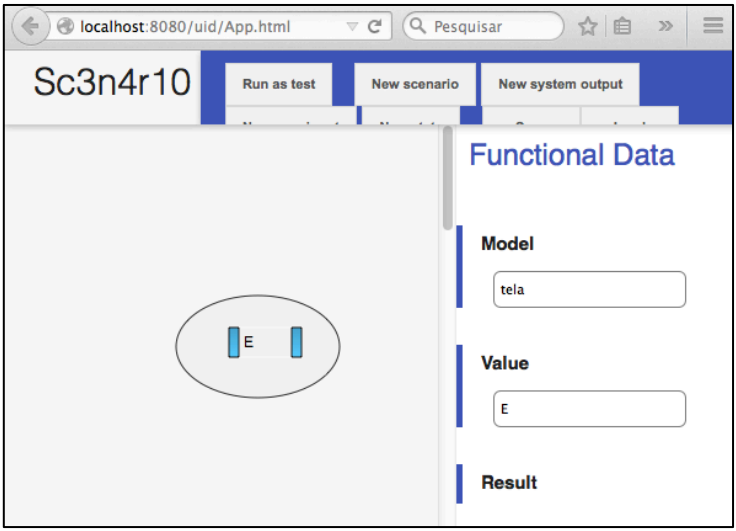
É importante salientar também que o US-UID representa o teste de aceitação e que ele próprio é executado. Durante a execução do US-UID o código da aplicação é chamado diretamente através da execução dos acessórios que representam o modelo conectado aos elementos dos US-UIDs, sem que seja necessário criar nenhum outro código de teste. É possível conectar diferentes entradas do usuário, ou saídas do sistema, ao mesmo modelo, que é representado pelo mesmo acessório que será executado.

6.7 US-UIDs ALTERNATIVOS

Esta dissertação aborda a construção de cenários de usuários ou US-UIDs como requisitos, sendo que cada cenário de usuário deve conter as expectativas ou intenção dos usuários. Deste modo, todos os cenários são expectativas explícitas dos usuários. Contudo, existem abordagens para especificação aonde os requisitos dos usuários são classificados como cenários de curso principal (*main course*) e cenários de cursos alternativos (*alternate courses*) (COCKBURN, 1997). Entretanto, um cenário de curso alternativo também pode ser visto como uma expectativa explícita do usuário sobre um requisito necessário que apresenta um resultado diferente daquele normalmente esperado, e, assim, é tratado como qualquer outro requisito e é expresso por um US-UID.

Para a calculadora é exemplificado a construção de um novo requisito aonde o resultado esperado é “E” ou um erro de operação. Assim é construído um estado de interação com a saída do sistema esperada de erro. A Figura 48 apresenta o estado de interação para construção do requisito do US-UID com erro de operação.

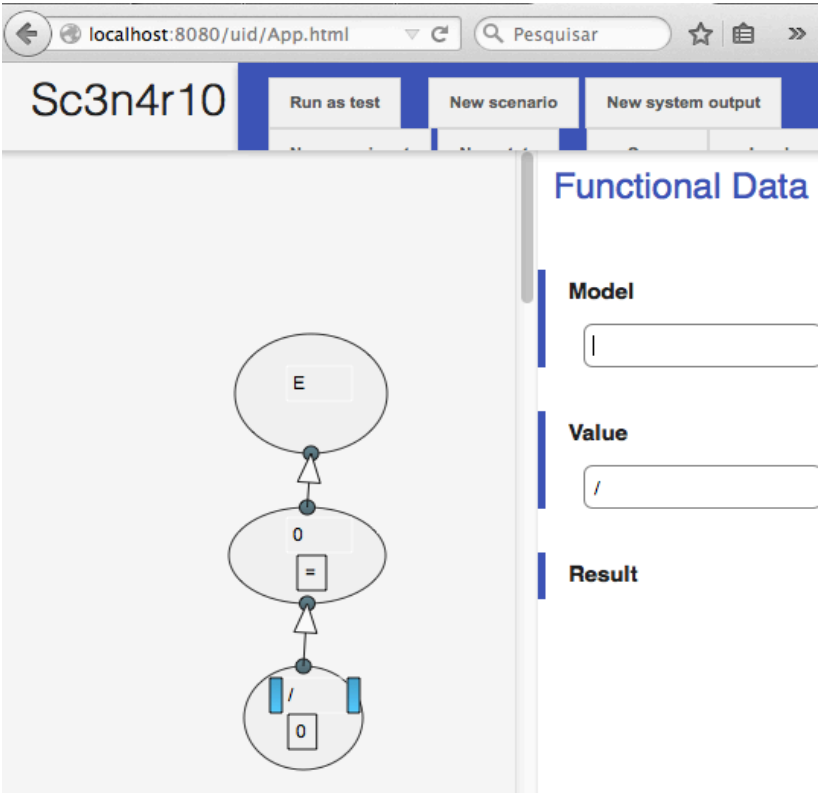
Figura 48 – Estado de interação para o US-UID de erro de operação



O US-UID apresentado na Figura 48 é incompleto, sendo que, não é possível compreender o erro de operação que o usuário deseja

representar. Assim, a Figura 49 apresenta o US-UID de erro de operação com as interações da operação de divisão por zero. Inicialmente são definidos os dados funcionais deste US-UID do mesmo modo utilizado para o US-UID da função de soma da calculadora (seção 6.3), ou seja, cada estado de interação é sincronizado com o estado de interação do UID da calculadora, as entradas de usuário são sincronizadas com a entrada ou botão do UID (Figura 42), e as saídas do sistema são sincronizadas com a saída do sistema tela do UID. Deste modo, não é necessária a edição do código do acessório, pois ele já foi anteriormente codificado no desenvolvimento do US-UID da função de soma da calculadora. Contudo, para implementação do código de aplicação recomenda-se a utilização de teste na própria linguagem do código fonte, um exemplo são caso de testes pelo *framework* JUnit.

Figura 49 - US-UID de erro de operação com as interações da operação de divisão por zero



Após a implementação do código de aplicação por outro *framework* de teste (por exemplo, JUnit), é possível executar o US-UID para verificar o resultado.

7 CONSIDERAÇÕES FINAIS

Um dos desafios entre as partes interessadas no desenvolvimento de aplicações é o entendimento e análise dos requisitos. No ATDD os AATs (Testes de Aceitação Automatizados) são adotados para promover a comunicação e colaboração entre as partes interessadas. Esta Dissertação aborda os US-UIDs para criação e execução automatizada de requisitos. A seção 7.1 apresenta algumas considerações sobre ATDD com US-UIDs e a seção 7.2 apresenta as conclusões.

7.1 CONSIDERAÇÕES SOBRE ATDD COM US-UIDS

Considerando que o ATDD é composto pelas práticas do TDD, este estudo é superficial e orienta-se apenas nos fundamentos de automação de testes e em algumas práticas de criação de testes. Deste modo, ainda resta uma lacuna entre os US-UIDs e práticas para ATDD. E também é evidente a falta de informações para determinar o uso de US-UIDs no processo evolutivo e/ou iterativo de desenvolvimento. Questões de pesquisas futuras podem envolver os seguintes pontos: criação de US-UIDs, avaliação de US-UIDs e o ritmo de desenvolvimento.

Sobre a criação dos US-UIDs, esta proposta considera que é responsabilidade dos usuários criar e entregar US-UID aos desenvolvedores. A ausência de US-UIDs indica falta de colaboração. Contudo, US-UIDs automatizados e com resultado de sucesso (verde) significam requisito cumprido.

Sobre a avaliação dos US-UIDs, a avaliação feita sobre o US-UID do *Jogo do 8* foi possível pois o experimento foi controlado. No entanto, determinar se o US-UID é um requisito válido ainda é um desafio pois, sobre a perspectiva do desenvolvedor, considera-se complicado fazer uma avaliação semântica sem a comunicação e decisão dos usuários. Por exemplo: Como resolver a ambiguidade dos US-UIDs da Figura 31 e da Figura 32? Ou como determinar se o US-UID é correto mas incompleto? Resolvido o problema de avaliação dos US-UIDs, então devem ser implementados todos os US-UIDs completos e corretos? Talvez não, muitos podem ser repetidos ou cópias.

Sobre o ritmo de desenvolvimento, este é, certamente, um assunto para futuras pesquisas, pois não há evidências claras quanto ao ritmo de desenvolvimento que deve ser assumido. O desenvolvimento de

pequenas partes evolutivamente pode ajudar na eliminação de problemas dos requisitos, como ambiguidade, e isso pode evitar grandes fatorações nos testes automatizados e no código. No entanto, muitas interações para troca de informações entre desenvolvimento e os usuários pode causar desperdícios de tempo e fadiga.

7.2 CONCLUSÕES

Esta Dissertação apresentou os US-UIDs e cumpriu com os objetivos em sua forma de avaliação, considerando a criação dos US-UIDs por usuários não técnicos em um estudo de caso; a implementação de um *framework* para execução dos US-UIDs; e avaliação da capacidade de rastreabilidade de códigos pelo *framework*.

A criação dos US-UIDs por usuários apresentou fatores de qualidade com 67% de completude e 90% de corretude no experimento realizado. Esses fatores de qualidade correspondem às expectativas dos usuários ou participantes sobre o *Jogo do 8*. No entanto, os requisitos do mundo real não são fáceis, sendo assim, é necessário assumir o risco sobre os fatores de qualidade dos requisitos ou implementar melhorias. Uma melhoria que pode ser adotada são os métodos de criação progressivos e regressivos. De acordo com a avaliação deste estudo, estes métodos apresentam resultados diferentes de fatores de qualidade. Então, enfatiza-se estes métodos para simplificar a criação de US-UIDs, indicando como melhor opção o método regressivo. Sendo que ter ciência sobre estes métodos pode ajudar a exploração do cenário de usuário e sua representação sobre os US-UIDs.

Este trabalho apresentou os US-UIDs e um *framework* para sua automação. Assim, os US-UIDs podem ser vistos em casos de teste automatizados que mantêm as propriedades de rastreabilidade, o que permite classificar a execução de um determinado caso de teste em sucesso, erro ou falha. Desta forma, os US-UIDs podem ser utilizados para determinar se o código de aplicação é adequado e cumpre os requisitos dos usuários.

A definição e avaliação dos US-UIDs foi publicada na *27th International Conference on Software Engineering and Knowledge Engineering* (SEKE 2015) (LONGO e VILAIN, 2015).

Este trabalho tem duas limitações importantes, que agora são ressaltadas para que possam ser abordadas em pesquisas futuras:

- Sobre experimento do *Jogo do 8*: o espaço amostral e o tamanho da amostra de participantes é considerado pequeno;

- Sobre a capacidade de rastreabilidade do *framework* proposto: pela execução dos US-UIDs, é possível rastrear erros, falhas e sucessos do código. Contudo, melhorias podem ser feitas, como por exemplo: rastrear e indicar a linha de código no editor deve ser importante.

REFERÊNCIAS

AGILE ALLIANCE. Disponível em: <http://guide.agilealliance.org/guide/acceptance.html>. Acesso em: 15 jun. 2015.

ALVESTAD, K. **Domain specific languages for executable specifications**. Norwegian University of Science and Technology Department of Computer and Information Science. 2007. 67p.

ARCURI, A.; IQBAL, M. Z.; BRIAND, L. Black-box system testing of real-time embedded systems using random and search-based testing. In: **Testing Software and Systems**. p. 95-110. Springer, 2010.

BECK, K. **Test-driven development:** by example. Addison-Wesley Professional, 2003.

BORG, R.; KROPP, M. Automated acceptance test refactoring. In: **Proceedings of the 4th Workshop on Refactoring Tools**. ACM, 2011. p. 15-21.

BUTLER, C. **You + Us + Them = Something Useful**. Disponível em: <https://www.newfangled.com/how-to-tell-the-users-story> Acesso em: 15 jun. 2015.

CISLAGHI, R.; WILGES, B.; NASSAR, S. M.; LONGO, D. H.; MATEUS, G. P. Avaliação de polos sob uma perspectiva georreferenciada**.** In: **XI Congresso Brasileiro de Ensino Superior a Distância** - ESUD. 2014.

Cockburn, A. **Structuring Use Cases with Goals1**. 1997.

CONNOLLY, D.; KEENAN, F.; MC CAFFERY, F. **Acceptance test-driven development by annotation of existing documentation**. 2010.

COWAN, A. **Your Best Agile User Story**. Disponível em: <http://www.alexandercowan.com/best-agile-user-story/>. Acesso em: 15 jun. 2015.

CRISPIN, L.; GREGORY, J. **Agile testing:** a practical guide for testers and agile teams**.** Pearson Education. 2009.

CUCUMBER. Disponível em: < https://cucumber.io/>. Acesso em: 15 jun. 2015.

DAMAS, C.; LAMBEAU, B.; DUPONT, P.; LAMSWEERDE, A. Generating annotated behavior models from end-user scenarios. In: **Software Engineering**, IEEE Transactions on. v. 31, n. 12, p. 1056-1073. IEEE. 2005.

DRUK, M; KROPP, M. ReFit: a fit test maintenance plug-in for the eclipse refactoring plug-in. In: **Developing Tools as Plug-ins (TOPI),** 2013 3rd International Workshop on. IEEE, p. 7-12. 2013.

EL-ATTAR, M; MILLER, J. Developing comprehensive acceptance tests from use cases and robustness diagrams. In: **Requirements Engineering**, v.15, n.3, pp. 285-306, Springer. 2010.

FITTER, M.; GREEN, TRG. When do diagrams make good computer languages?. In: **International Journal of man-machine studies**. Elsevier, v.11, n.2, p.235—261. 1979.

GÄRTNER, M. **ATDD by example:** a practical guide to acceptance test-driven development. Addison-Wesley, 2012.

GIUFFRA, C.; VILAIN, P. Modelagem da interação do usuário no desenvolvimento ágil. In: **SULCOMP**, v. 5, n. 1, 2010.

GRANJA, A. D. Disponível em: <http://www.fec.unicamp.br/~adgranja/index_arquivos/preparacao_definicao_teses.pdf> Acesso: jun. 2015.

GREILER, M.; ZAIDMAN, A.; VAN DEURSEN, A.; STOREY, M. Strategies for avoiding text fixture smells during software evolution. In: **Mining Software Repositories (MSR)**, 2013 10th IEEE Working Conference on. IEEE. p. 387-396. 2013.

GUELL, N.; SCHWABE, D.; VILAIN, P. Modeling interactions and navigation in web applications. In: **Conceptual Modeling for E- Business and the Web. Springer**, p. 115-127. 2000.

HANSSEN, G. K.; HAUGSET, B. Automated acceptance testing using fit. In: **System Sciences**, 2009. HICSS'09. 42nd Hawaii International Conference on, pp.1-8. IEEE. 2009.

HAUGSET, B.; HANSSEN, G. K. Automated acceptance testing: a literature review and an industrial case study. In: **Agile**, 2008. AGILE'08. Conference. IEEE, p. 27-38. 2008.

HAYES, J. H.; DEKHTYAR, A.; JANZEN, D. S. Towards traceable test-driven development. In: **Traceability in Emerging Forms of Software Engineering**, 2009. TEFSE'09. ICSE Workshop on. IEEE, 2009. p. 26-30.

HÜTTERMANN, M. **Agile ALM**. Manning, 2011.

IEEE Recommended Practice for Software Requirements Specifications. In: **IEEE Std 830-1998**, p. 1-40, 1998.

INTERNATIONAL ORGANIZATION FOR STANDARDIZATION. **ISO/IEC/IEEE 24765:2010** - Systems and software engineering – Vocabulary. ISO/IEC/IEEE, 24765. 2010.

KAMALRUDIN, M. Automated software tool support for checking the inconsistency of requirements. In: **Automated Software Engineering**, 2009. ASE'09. 24th IEEE/ACM International Conference on. IEEE. p. 693-697. 2009.

KAMALRUDIN, M.; GRUNDY, J. Generating essential user interface prototypes to validate requirements. In: **Proceedings of the 2011 26th IEEE/ACM International Conference on Automated Software Engineering**. IEEE Computer Society. p. 564-567. 2011.

KAMALRUDIN, M.; GRUNDY, J.; HOSKING, J.: Tool support for essential use cases to better capture software requirements. In: **Proceedings of the IEEE/ACM international conference on Automated software engineering**. ACM. p. 255—264. 2010.

KAMALRUDIN, M.; SIDEK, S.; AIZA, M. N.; ROBINSON, M. **Automated acceptance testing tools evaluation in agile software development**. Pakistan: Publications International Lahore, 2013.

LEHMANN, E. L.; ROMANO, J. P. **Testing statistical hypotheses**. Springer, 2006.

LIEBEL, G.; ALÉGROTH, E.; FELDT, R. State-of-Practice in GUI-based System and Acceptance Testing: An Industrial Multiple-Case Study. In: **Software Engineering and Advanced Applications (SEAA), 2013 39th EUROMICRO Conference on**. IEEE. p. 17-24. 2013.

LONGO, D. H., VILAIN, P. **Creating user scenarios through user interaction diagrams by non-technical customers**. In: The 27th International Conference on Software Engineering and Knowledge Engineering – SEKE. KSI Research Inc. and Knowledge Systems Institute Graduate School, pp.330-335. 2015.

LONGO, D. H.; WILGES, B.; VILAIN. P.; CISLAGHI, R. Fixture Setup through Object Notation for Implicit Test Fixtures. In: **Aceito: Journal of Computer Science** - JCS. 2015.

LUCIA, A. D.; QUSEF, A. Requirements engineering in agile software development. In: **Journal of Emerging Technologies in Web Intelligence**, v. 2, p. 212-220. 2010.

MASSEY, A. K.; RUTLEDGE, R. L.; ANTON, A. I.; SWIRE, P. P. Identifying and classifying ambiguity for regulatory requirements. In: **Requirements Engineering Conference (RE)**, 2014 IEEE 22nd International, IEEE, pp. 83-92. 2014.

MEINKE, K.; NIU, F. A **Learning-based approach to unit testing of numerical software**. In: **ICTSS.** p. 221-235. Springer. 2010.

MILLER, R., COLLINS, C. T. **Acceptance testing**. Proc. XPUniverse, 2001.

NAKAGAWA, H; TAGUCHI, K; HONIDEN, S. Formal specification generator for KAOS: model transformation approach to generate formal specifications from KAOS requirements models. In: **Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering,** ACM, p. 531-532. APA. 2007.

PAPADAKIS, M.; MALEVRIS, N.; KALLIA, M. Towards automating the generation of mutation tests. In**: Proceedings of the 5th Workshop on Automation of Software Test,** p. 111-118. ACM. 2010.

PARK, S.; MAURER, F. **An extended review on story test driven development**. In: University of Calgary. 2010.

POPOVIC, T. Concordion. Fonte: Disponível em: <https://www.academia.edu/7698408/Concordion>. Acesso em: 15 jun. 2015.

PREECE, J.; ROGERS, Y.; SHARP, H.; BENYON, D.; HOLLAND, S.; CAREY, T. **Human-computer interaction**. ADDISON-WESLEY LONGMAN LTD, 1994.

SAMADHIYA, D.; RANJAN, A. AcceptSoftware: A Tool for Executable Acceptance Test Driven Development. In: **Global Journal of Business Management and Information Technology**. v.1, n.2, pp. 131—139. 2011.

SPARX SYSTEMS. **Writing use case scenarios for model driven development**. 2010.

SOMMERVILLE. I. **Software Enginnering**. 9th ed, p. 773. Addison-Wesley, 2011.

THONGTANUNAM, P., TANTITHAMTHAVORN, C., KULA, R. G., YOSHIDA, N., IIDA, H., MATSUMOTO, K. Who should review my code? A file location-based code-reviewer recommendation approach for Modern Code Review. In: **Software Analysis**, Evolution and Reengineering (SANER), 2015 IEEE 22nd International Conference on, p. 141-150. IEEE. 2015.

VALDERAS, P.; PELECHANO, V. A survey of requirements specification in model-driven development of web applications. In: **ACM Transactions on the Web (TWEB)**, v. 5, n. 2, p. 10. 2011.

VILAIN, P. **Modelagem da interação com o usuário em aplicações hipermídia**. 2002. Tese (Doutorado)- PUC-RIO, Rio de Janeiro, 2002.

VILAIN, P.; SCHWABE, D.; SOUZA, C. S. **A Diagrammatic Tool for Representing User Interaction in UML**. UML 2000: p. 133-147. 2000.

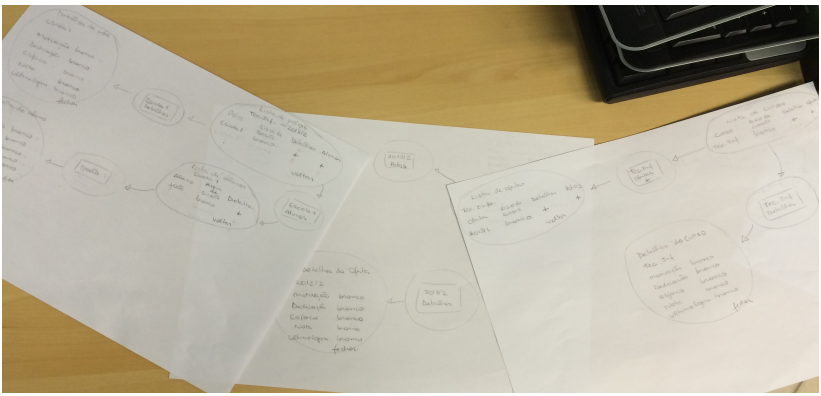
WAZLAWICK, R. S. **Análise e projeto de sistemas da informação Orientado a Objetos**. Rio de Janeiro: Elsevier Brasil, 2004.

YIN, R. K. **Case study research: design and methods**, 3rd ed., res. Applied Social Research Methods Series, L. Bickman and D. J. Rog, Eds. Sage Publications, v. 5. 2003.

ZEFERINO, N. V.; VILAIN, P. A model-driven approach for generating interfaces from user interaction diagrams. In: **Proceedings of the 16th International Conference on Information Integration and Web-based Applications & Services**. ACM, p. 474-478. 2014.

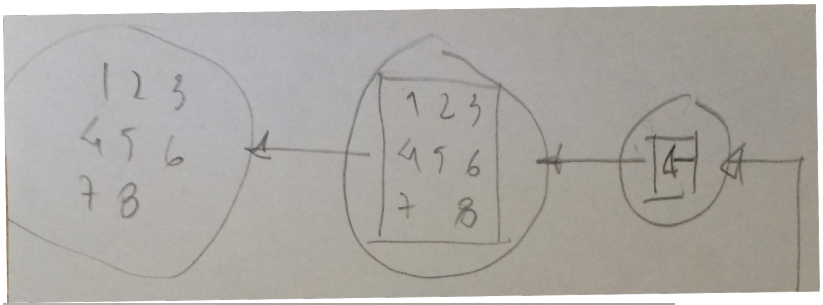
8 APÊNDICE

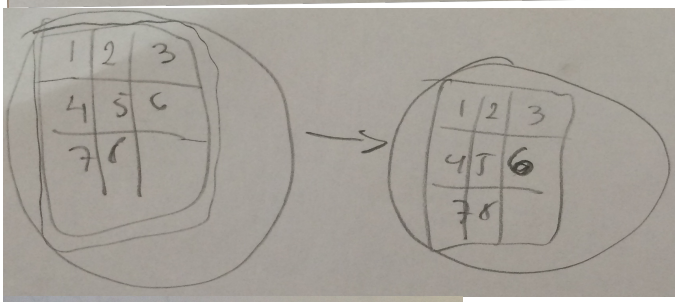
APÊNDICE A - FOTO DO US-UIDS DO MÓDULO SISTEMA WEB

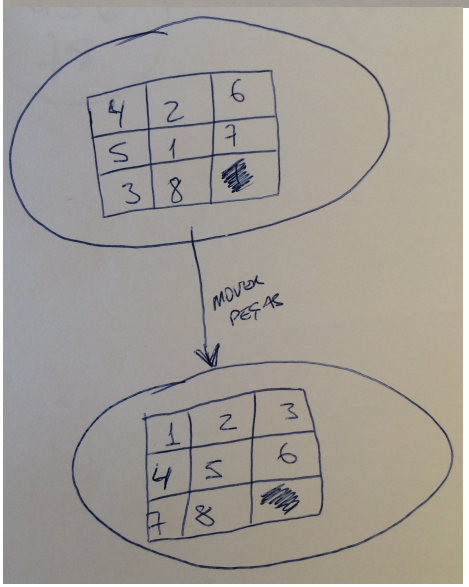


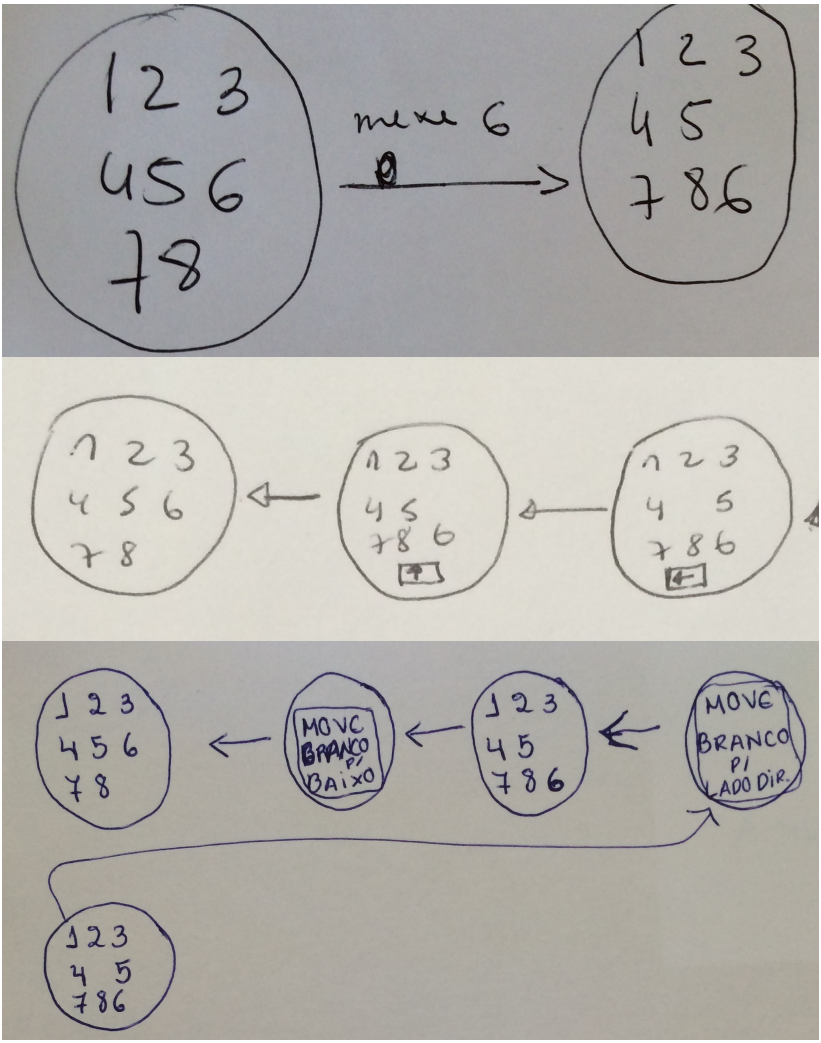
Fonte: Elaborado pelo autor

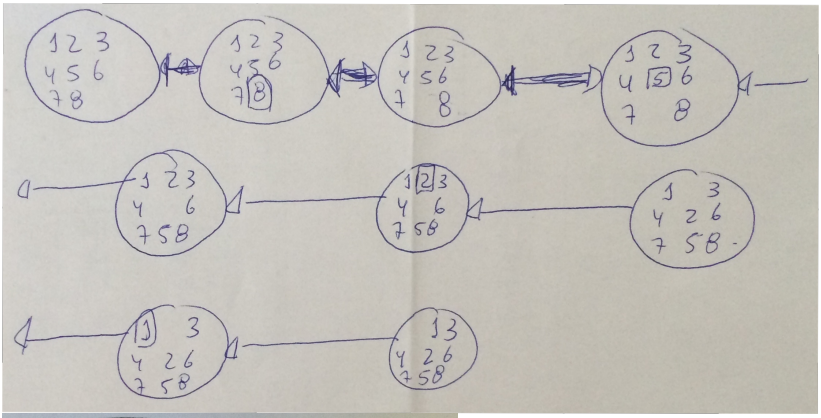
APÊNDICE B – FOTOS DOS US-UIDS DO EXPERIMENTO II

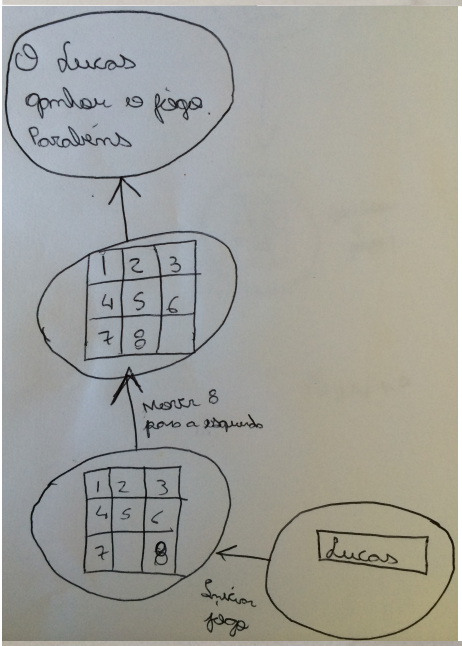


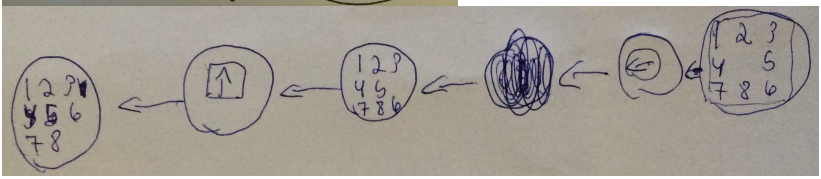


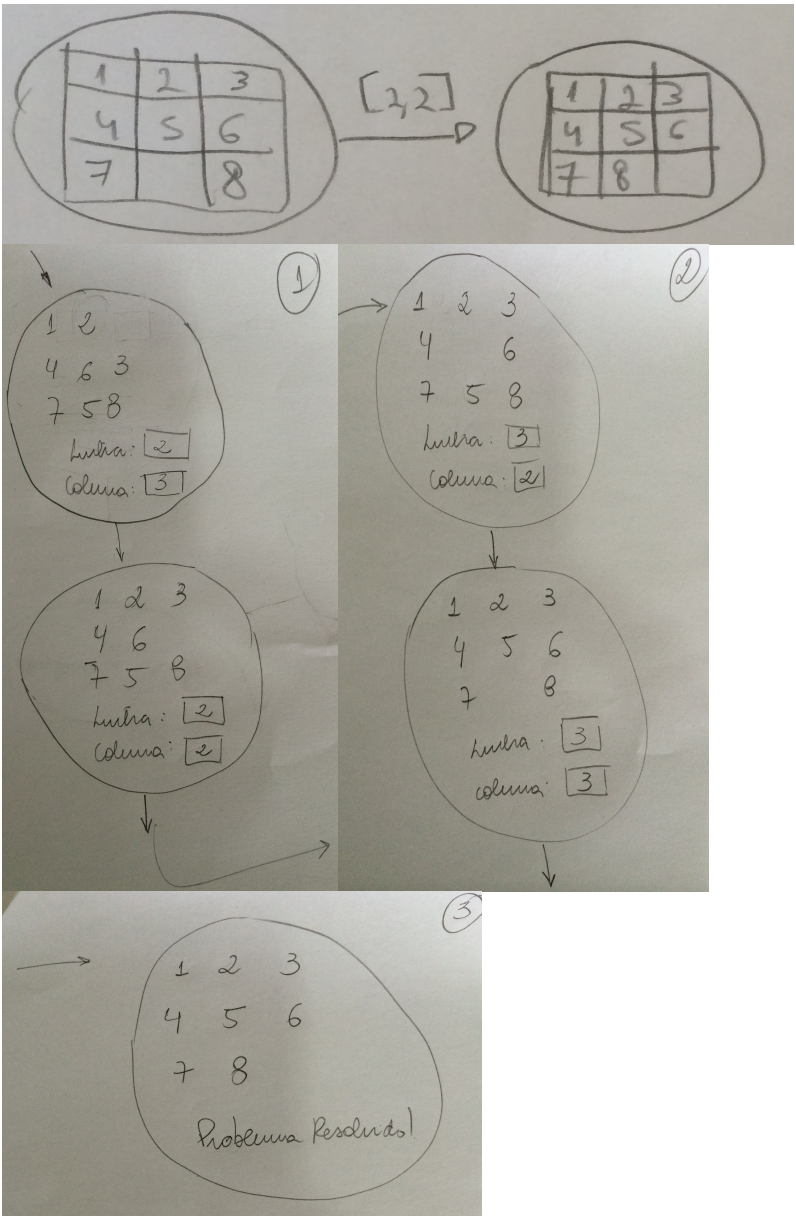


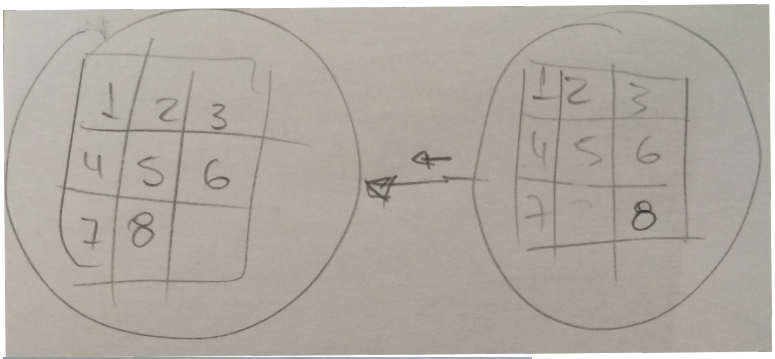

1 2 3
4 5 6
7 8
mexe 6
1 2 3
4 5
7 8 6
MOVE BRANCO P/ BAIXO
MOVE BRANCO P/ LADO DIR.

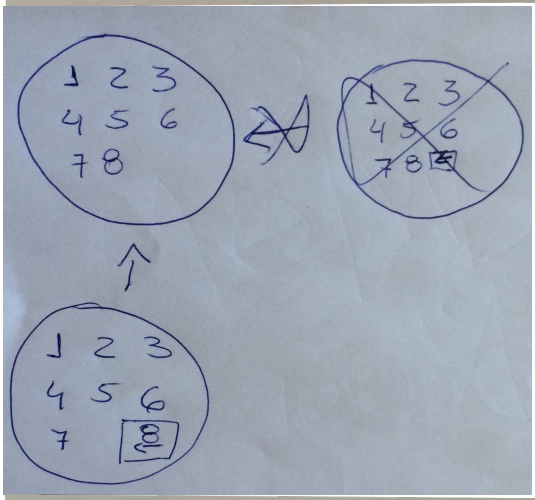


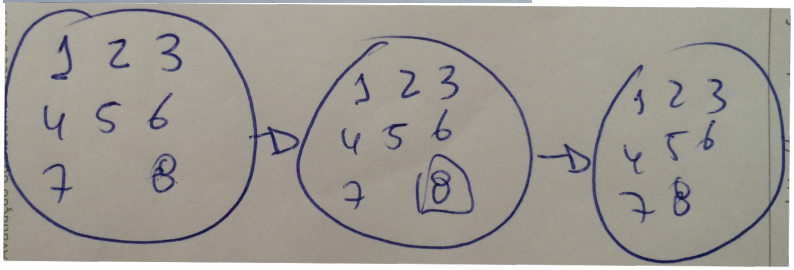

O Lucas
ganhou o jogo.
Parabéns
Mover 8
para a esquerda
Lucas
Jogador
jogo




1 2 3
4 5 6
7 8
[2,2]
1 2 3
4 5 6
7 8
1
1 2
4 6 3
7 5 8
Linha: 2
Coluna: 3
1 2 3
4 6
7 5 8
Linha: 2
Coluna: 2
2
1 2 3
4 6
7 5 8
Linha: 3
Coluna: 2
1 2 3
4 5 6
7 8
Linha: 3
coluna: 3
3
1 2 3
4 5 6
7 8
Problema Resolvido!


1 2 3
4 5 6
7 8
1 2 3
4 5 6
7 8


1 2 3
4 5 6
7 8
1 2 3
4 5 6
7 8
1 2 3
4 5 6
7 8


1 2 3
4 5 6
7 8
1 2 3
4 5 6
7 8
1 2 3
4 5 6
7 8

ANEXO A - JOGO DO 8

O *Jogo do 8* consiste em um tabuleiro de 3 linhas por 3 colunas. O tabuleiro é preenchido com a sequência de números de 1 a 8 e um espaço vazio. O objetivo do jogo é, partindo de um estado aleatório, ordenar a sequência de algarismos. A Figura 50 apresenta o seu estado final.

Figura 50 - Estado final do *Jogo do 8*

1	2	3
4	5	6
7	8	

As operações para configuração do tabuleiro podem ser representadas como: mover o espaço em branco para cima, para a esquerda, para a direita ou para baixo. Nas implementações digitais, o espaço vazio é movimentado usando as teclas de direção do teclado ou o *click* no *mouse* sobre um algarismo vizinho ao espaço vazio.